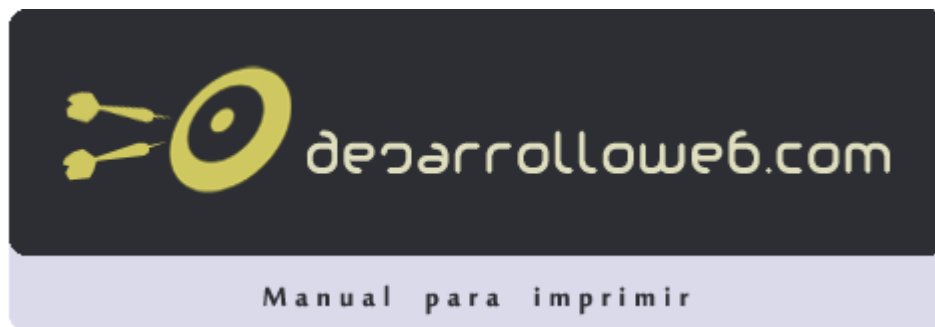


# Sistema de autenticación PHP



## Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

### **Miguel Angel Alvarez**

Director de DesarrolloWeb.com  
<http://www.desarrolloweb.com>  
(8 capítulos)

### **Eugenia Bahit**

Desarrolladora ASP y PHP  
<http://www.cmzk.com.ar>  
(2 capítulos)

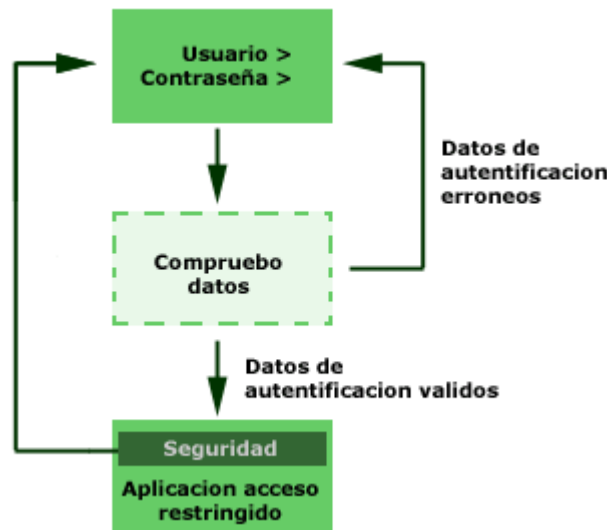
## Funcionamiento del sistema de autenticación en PHP

Un sistema de autenticación es un módulo de seguridad para asegurarnos de que el usuario que visita las páginas es quien dice ser. Por supuesto, sabiendo que ese usuario es conocido, podremos darle acceso a más aspectos de la página que si fuese un usuario desconocido. Pero supongo que, si estás leyendo este artículo, ya conocerás lo que es un sistema de autenticación y lo que deseas hacer es crear uno para tus páginas.

**Referencia:** Este manual requiere el conocimiento básico de PHP que, en caso de no tenerlo, se puede aprender en nuestro [Manual de Programación en PHP](http://www.desarrolloweb.com/manuales/37/).

### Esquema de un sistema de autenticación

Vamos a empezar por definir un diagrama para realizar la autenticación de usuario en unas páginas web, que nos servirá para programar luego las páginas ajustándose al diagrama.



En la imagen anterior podemos ver el diagrama, que empieza por la página donde se pide un usuario y contraseña para acceder a la aplicación de acceso restringido.

Los datos de autenticación (usuario y contraseña escritos en la página inicial) se envían a la página dibujada con línea de puntos, que se encarga de hacer una comprobación de dichos datos del usuario. Según los datos de autenticación, se redirecciona al navegador a la página de la aplicación restringida, en caso de que sean correctos, o a la página donde volver a escribir el usuario/contraseña, en caso de que sean incorrectos. Esta página la he dibujado con línea de puntos porque no es una página donde se pare el navegador para nada, sino que sólo es una página de paso que redirecciona a un sitio u otro dependiendo de los datos que reciba.

La aplicación de acceso restringido, aparte de mostrar las funcionalidades que queríamos proteger con usuario contraseña, debe de realizar unas comprobaciones de seguridad para saber si se ha pasado con éxito el proceso de autenticación o si se está intentando acceder de manera no permitida a esa página. Esta comprobación la he dibujado como una capa con color verde más oscuro sobre la página de la aplicación. Si no se satisface dicha comprobación (el usuario no se ha autenticado correctamente) se vuelve a la página donde escribir el usuario y la contraseña.

Este es el esquema básico, que espero que se entienda bien. Ahora, veamos algunas preguntas que podría hacerse el lector.

### **¿Por qué hacemos esta comprobación de seguridad dentro de la aplicación?**

Podría ser que alguien conociese la URL de la aplicación de acceso restringido y la escribiese directamente sobre la barra de direcciones del explorador, así que hacemos esta comprobación para saber que realmente no se está accediendo sin pasar por la página que comprueba si el usuario/contraseña es correcto

### **¿Cómo sabemos que ciertamente se ha pasado por la página que comprueba los datos de autenticación?**

Esta comprobación la podríamos hacer de varias maneras, así pues, depende de nuestro script de autenticación y el nivel de seguridad que tratemos de implementar. Un ejemplo simple podría ser crear una variable de sesión en la página que comprueba los datos, si es que eran correctos, y en capa de seguridad de las páginas de acceso restringido comprobaríamos si esa sesión está o no definida.

### **En caso de desear burlar la seguridad, ¿Cómo podría un usuario entrar a la página de la aplicación si no hay enlaces directos y para pasar a ella necesitamos que nos redirija la página de comprobación del usuario/contraseña?**

Pues de diversas maneras, para empezar, el historial de los ordenadores guarda las URL a las que se ha accedido y cualquier persona podría recuperar la URL de nuestra aplicación con acceso restringido. También se podría probar distintas URL que podríamos imaginarnos como posibles para la aplicación y esperar a acertar con el nombre de archivo en algún momento, incluso esta tarea se la podríamos encomendar a un programa para realizar muchas más pruebas. En cualquier caso, nuestra seguridad no se puede quedar en simplemente que los posibles intrusos no conozcan la dirección de la página.

**[Ver el sistema en funcionamiento](#)**

*Artículo por **Miguel Angel Alvarez***

## ***Página inicial con el formulario de autenticación en PHP***

Vamos a realizar la página inicial, que tiene el formulario de autenticación en el que el visitante debería rellenar con su usuario y contraseña. Como es la página inicial, la llamaremos index.php, que es el documento por defecto configurado en nuestro servidor.

Para realizar esta página, utilizaremos HTML básico, excepto en una comprobación que nos permitirá saber si se accede al formulario de nuevo por no haber introducido correctamente el usuario y contraseña, pues, en ese caso, habría que mostrar un cartelito informando que el usuario o la contraseña no son válidos.

Para pasar a la página inicial el mensaje de que el usuario/contraseña introducidos no son válidos utilizaremos una variable pasada a través de la URL. La llamaremos errorusuario, y si contiene la cadena "si" es que estamos recibiendo un error.

El código sería el siguiente:

```
<html>
```

```
<head>
<title>Autenticación PHP</title>
</head>
<body>
<h1>Autenticación PHP</h1>
<form action="control.php" method="POST">
<table align="center" width="225" cellspacing="2" cellpadding="2" border="0">
<tr>
<td colspan="2" align="center">
<?if ($_GET["errorusuario"]=="si"){?>
bgcolor=red><span style="color:ffffff"><b>Datos incorrectos</b></span>
<?}else{?>
bgcolor=#cccccc>Introduce tu clave de acceso
<?}?></td>
</tr>
<tr>
<td align="right">USER: </td>
<td><input type="Text" name="usuario" size="8" maxlength="50"></td>
</tr>
<tr>
<td align="right">PASSWD: </td>
<td><input type="password" name="contrasena" size="8" maxlength="50"></td>
</tr>
<tr>
<td colspan="2" align="center"><input type="Submit" value="ENTRAR"></td>
</tr>
</table>
</form>
</body>
</html>
```

**Nota:** La variable errorusuario, recibida por la URL y que informa si se produjo un error anterior al introducir usuario y contraseña, se está recogiendo por mediación del array asociativo \$\_GET, que guarda todas las variables enviadas por la URL. Aprendemos más sobre pasar variables por la URL en la dirección: <http://www.desarrolloweb.com/articulos/317.php>

El formulario tiene el atributo action dirigido hacia la página "control.php", que es la que se encarga de recoger los datos y ver si son correctos. Será tratada en el próximo capítulo.

### [Ver el sistema en funcionamiento](#)

*Artículo por **Miguel Angel Alvarez***

## **Control de los datos de autenticación en PHP**

Esta página será encargada de decidir si los datos de configuración son correctos y actuar en consecuencia. Dependiendo del nivel de seguridad que queramos aplicar a nuestra aplicación, esta página será más o menos complicada.

En un principio no deseo liar mucho las cosas, así que explicaré una versión muy reducida de este archivo de control, en la que se comprueba si el usuario y contraseña sean dos valores específicos. Esto tiene la desventaja que sólo podemos crear un usuario/contraseña distinto y no un sistema que permita muchos usuarios distintos. Bueno, en realidad si que permitirá que accedan muchos usuarios a la vez, pero utilizando todos el mismo nombre de usuario y contraseña.

En aplicaciones más avanzadas podríamos tener en una base de datos una lista de usuarios

con sus contraseñas. Entonces, en este archivo de control deberíamos hacer una búsqueda para ver si existe una correspondencia en la base de datos de ese usuario con esa contraseña. Esto lo veremos en adelante, ahora nos quedamos con la versión reducida.

Después de la comprobación podrán pasar dos cosas:

Si los datos son correctos, definirá una variables de sesión que servirá para saber que ese visitante ha sido validado correctamente y tiene permiso para acceder a la aplicación. Además redireccionará al visitante a la página de la aplicación restringida.

Si el usuario/contraseña no era correcto, se envía al navegador a la página de inicio pasando la variable `errorusuario=si`, que indica que ha habido un error en la autenticación.

El código se puede ver a continuación:

```
<?
//vemos si el usuario y contraseña es válido
if ($_POST["usuario"]=="miguel" && $_POST["contrasena"]=="qwerty"){
    //usuario y contraseña válidos
    //defino una sesion y guardo datos
    session_start();
    $_SESSION["autenticado"]="SI";
    header("Location: aplicacion.php");
}else {
    //si no existe le mando otra vez a la portada
    header("Location: index.php?errorusuario=si");
}
?>
```

**Referencia:** Dos enlaces a documentación relacionada en nuestro manual de Programación en PHP:  
Aprender el uso de sesiones <http://www.desarrolloweb.com/articulos/320.php?manual=12>

Aprender el paso de variables por formulario <http://www.desarrolloweb.com/articulos/318.php?manual=12>

**[Ver el sistema en funcionamiento](#)**

*Artículo por **Miguel Angel Alvarez***

## Capa de seguridad en PHP

Este archivo, en nuestro caso llamado seguridad.php, se encargará de dotar seguridad a toda la aplicación de acceso restringido. La técnica que vamos a utilizar es incluirlo al principio de todas las páginas que queramos que permitan un acceso restringido.

El módulo de seguridad, incluido al principio de cada archivo, realizará las comprobaciones oportunas y actuará permitiendo ver el archivo o denegando su visualización dependiendo de dichas comprobaciones.

Dependiendo del nivel de seguridad que deseemos implementar, la creación de este archivo puede ser más o menos complicada. Como no deseo complicar en un principio los scripts, esta versión resultará bastante sencilla.

Lo único que haré será recuperar la variable de sesión donde guardo si ese usuario ha sido autenticado o no. Luego se comprueba esa variable para saber si se ha autenticado el usuario o no, realizando estas acciones:

Si no se había autenticado, redirijo al navegador a la página que tiene el formulario de autenticación. Además, salgo del script PHP, con lo que la página deja de ejecutarse y el resto no se verá. Sólo se mandará al navegador la redirección con lo que el navegador se moverá la formulario y será imposible ver nada en la página segura.

Si se había autenticado, no hago nada. Ni tan siquiera trato este caso, de modo que se seguiría ejecutando la página con el contenido que correspondiese. No hay que olvidar que este archivo de seguridad se va a ejecutar como un include al principio de todos los archivos de la aplicación restringida, lo que significa que, si no se hace nada, se seguiría mostrando la página donde este archivo está incluido.

El código se puede ver a continuación:

```
<?
//Inicio la sesión
session_start();

//COMPRUEBA QUE EL USUARIO ESTA AUTENTICADO
if ($_SESSION["autenticado"] != "SI") {
    //si no existe, envío a la página de autenticacion
    header("Location: index.php");
    //ademas salgo de este script
    exit();
}
?>
```

### [Ver el sistema en funcionamiento](#)

Artículo por **Miguel Angel Alvarez**

## **Archivos de la aplicación con acceso restringido en PHP**

La aplicación con acceso restringido se realizará como cualquier otra aplicación de PHP, con la salvedad de que, a todos los archivos que queramos proteger, habrá que incluirles al principio la capa de seguridad, representada por el archivo seguridad.php.

Como decía, todo en el archivo de la aplicación se realizará como cualquier otro archivo de PHP, es decir, con sólo incluir el módulo de seguridad, el archivo ya tendrá el acceso restringido y todo lo demás lo haremos de manera transparente a este estado de seguridad.

El código de una página segura sería el siguiente:

```
<?include ("seguridad.php");?>
<html>
<head>
<title>Aplicación segura</title>
</head>
<body>
<h1>Si estás aquí es que te has autenticado</h1>
<br>
----
<br>
Aplicación segura
```

```
<br>
----
<br>
<br>
<a href="salir.php">Salir</a>
</body>
</html>
```

**Importante:** El include del archivo seguridad.php se ha de realizar en la primera línea del archivo PHP de la aplicación. Si no lo hacemos en la primera línea o si escribimos texto en la página antes de incluir la capa de seguridad, el script podría fallar y hacer que no funcione la aplicación o que sea menos segura. Este efecto se produce porque no se puede escribir en la página nada si se desea hacer una redirección con PHP (función header) y si se escribe algo, la redirección no podrá funcionar.

Un detalle que hemos incluido es un enlace para salir de la aplicación, que se dirige a el archivo salir.php. Explicaremos el sentido de esta acción y el script de salir.php en el siguiente capítulo.

### [Ver el sistema en funcionamiento](#)

*Artículo por **Miguel Angel Alvarez***

## **Salir de la aplicación segura en PHP**

La seguridad de la aplicación se basa en la definición de unas variables de sesión que se consultan en cada página segura. Puede ocurrir que el usuario entre en la aplicación e inicie una sesión y que se marche de la aplicación segura sin cerrar la sesión, con lo que quedaría abierta para que cualquier otra persona pueda acceder a la aplicación volviendo por el historial de páginas del navegador.

Las sesiones se finalizan solas cuando pasa un determinado tiempo sin recibir nuevas peticiones, pero no deseamos que antes de que se finalicen se pueda acceder con ese ordenador a nuestra aplicación restringida.

Parece interesante, pues, ofrecer al visitante la opción de acabar la sesión en cualquier momento, para asegurarnos en ese caso que la sesión se ha terminado y no se podrá acceder si no es introduciendo nuevamente el usuario y contraseña correctos.

El archivo en concreto lo único que hace es terminar la sesión asociada a su acceso. Podemos ver el código a continuación.

```
<?
session_start();
session_destroy();
?>
<html>
<head>
<title>Has salido!!</title>
</head>
<body>
Gracias por tu acceso
<br>
<br>
<a href="index.php">Formulario de autenticación</a>
```

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

## [Ver el sistema en funcionamiento](#)

*Artículo por Miguel Angel Alvarez*

## **Diferentes formas de cerrar sesión en PHP**

Veremos como cerrar la sesión del usuario cuando:

- El tiempo de inactividad del usuario supere "x" cantidad de tiempo (segundos, minutos, etc...).
- El usuario cierre el navegador y abandone por completo nuestro sitio.

### **Cierre de sesión por inactividad en PHP: módulo de control de datos**

Algo que puede parecer muy obvio para unos y muy complejo para otros, pero que innegablemente muchos de nosotros nos hemos preguntado en algún momento: ¿Cómo caducar una sesión en PHP?

Ahora veremos que tan sencillo es. Solo tendremos que:

- Crear una nueva sesión que guarde una fecha y hora
- Comprobar en nuestra capa de seguridad el tiempo transcurrido entre la sesión guardada y la hora actual
- Actualizar la sesión o destruirla según corresponda

Lo primero que debemos hacer entonces, es crear la nueva sesión y asignarle como valor, la hora actual. Esto lo haremos en el momento que el usuario ingresa al sistema con sus datos de acceso.

```
<?
//vemos si el usuario y contraseña es válido
if ($_POST["usuario"]=="miguel" && $_POST["contrasena"]=="qwerty"){
    //usuario y contraseña válidos
    session_name("loginUsuario");
    //asigno un nombre a la sesión para poder guardar diferentes datos
    session_start();
    // inicio la sesión
    $_SESSION["autenticado"]="SI";
    //defino la sesión que demuestra que el usuario está autorizado
    $_SESSION["ultimoAcceso"]= date("Y-n-j H:i:s");
    //defino la fecha y hora de inicio de sesión en formato aaaa-mm-dd hh:mm:ss
    header ("Location: aplicacion.php");
}else {
    //si no existe le mando otra vez a la portada
    header("Location: index.php?errorusuario=si");
}
?>
```

### **Cierre de sesión por inactividad en PHP: módulo de seguridad**

El segundo paso, será comprobar el tiempo transcurrido entre la fecha guardada y la hora actual en nuestra capa de seguridad y actuar en consecuencia.



Para hacerlo, tendremos que realizar un cálculo muy sencillo:

tiempo transcurrido = (hora actual - fecha guardada)

Y luego, restará saber si el tiempo transcurrido es mayor, menor o igual que el tiempo de caducidad de la sesión (representado como "x"):

si (tiempo transcurrido >= x), actúo en consecuencia a lo hallado

Para efectuar estos cálculos utilizaremos como unidad de tiempo el segundo. En nuestro ejemplo, caducaremos la sesión, transcurridos 10 minutos de inactividad (donde:  $10 \times 60 = 600$  segundos). Para efectuar estos cálculos y tomar como unidad de medida el segundo, será necesario convertir las fechas a segundos. Para ello, utilizaremos la función `strtotime`.

Por lo tanto, calcularemos el tiempo transcurrido (tiempo transcurrido = (hora actual - fecha guardada)) de la siguiente manera:

```
<?
//iniciamos la sesión
session_name("loginUsuario");
session_start();

//antes de hacer los cálculos, compruebo que el usuario está logueado
//utilizamos el mismo script que antes
if ($_SESSION["autenticado"] != "SI") {
    //si no está logueado lo envío a la página de autenticación
    header("Location: index.php");
} else {
    //sino, calculamos el tiempo transcurrido
    $fechaGuardada = $_SESSION["ultimoAcceso"];
    $ahora = date("Y-n-j H:i:s");
    $tiempo_transcurrido = (strtotime($ahora)-strtotime($fechaGuardada));

    //comparamos el tiempo transcurrido
    if($tiempo_transcurrido >= 600) {
        //si pasaron 10 minutos o más
        session_destroy(); // destruyo la sesión
        header("Location: index.php"); //envío al usuario a la pag. de autenticación
        //sino, actualizo la fecha de la sesión
    } else {
        $_SESSION["ultimoAcceso"] = $ahora;
    }
}
?>
```

*Artículo por **Eugenia Bahit***

## **Cierre de sesión al cerrar el navegador en PHP**

### **Introducción**

Mucho más sencillo de lo que uno imagina, es hacer que una sesión caduque en forma automática cuando el usuario cierre el navegador.

En principio, vamos a definir a que nos referimos con cerrar el navegador.

El servidor entenderá que el usuario cerró el navegador cuando ya no se encuentre visitando ninguna de las páginas de nuestro sitio. Es decir, si un usuario que para navegar nuestro sitio abrió al menos 2 páginas en 2 ventanas diferentes, el servidor considerará que cerró el navegador cuando hasta la última ventana sea abandonada. Ya sea porque el usuario la cerró o porque fue hacia otro sitio que no es el nuestro.

Entonces, para que la sesión caduque al cerrar el navegador, habrá que, por un lado, forzar al php.ini a que propague la sesión solamente en cookies y por otro lado, asignarle a ésta, una duración cero.

Para forzar al php.ini hay dos formas: modificar el php.ini directamente o cambiar los valores desde nuestro script.

### Opción 1

Configurar el archivo de inicio de php (php.ini) en forma directa.

Si tienes acceso a este archivo, habrá que buscar y cambiar el valor a:

```
session.use_trans_sid = 0  
session.use_only_cookies = 1
```

Esta última, justamente será la que indicará que la sesión debe propagarse solo a través de cookies.

### Opción 2

Cambiar la configuración del php.ini desde nuestro script php.

Esta opción consiste en forzar al php.ini desde nuestro script php (No todos los servidores tienen habilitada esta opción. Si el servidor no es propio, por las dudas, consulta con tu proveedor).

Para hacerlo, utilizaremos la función `ini_set()`

```
ini_set("session.use_trans_sid","0");  
ini_set("session.use_only_cookies","1");
```

Ahora solo restará, cambiar el parámetro de duración a la cookie de la sesión. Esto lo haremos en nuestro script con la siguiente instrucción:

```
session_set_cookie_params(0, "/", $_HTTP_SERVER_VARS["HTTP_HOST"], 0);
```

Con lo cual estaremos indicando una duración de 0 (cero) segundos. Esto significará que durará hasta que termine el script.

Por fin, hemos llegado al código:

### Cierre de sesión al cerrar el navegador en PHP: módulo de control de datos

Verás reflejados los cambios al script anterior, indicados en **negritas**. Estos cambios serán también los que se apliquen al módulo de seguridad. De todas formas, aquí te cargo los dos códigos.

<?

```
//si es necesario cambiar la config. del php.ini desde tu script
ini_set("session.use_only_cookies","1");
ini_set("session.use_trans_sid","0");

//vemos si el usuario y contraseña es válido
if ($_POST["usuario"]=="miguel" && $_POST["contrasena"]=="qwerty"){
    //usuario y contraseña válidos
    session_name("loginUsuario");
    //asigno un nombre a la sesión para poder guardar diferentes datos
    session_start();
    // inicio la sesión
    session_set_cookie_params(0, "/", $HTTP_SERVER_VARS["HTTP_HOST"], 0);
    //cambiamos la duración a la cookie de la sesión
    $_SESSION["autenticado"] = "SI";
    //defino la sesión que demuestra que el usuario está autorizado
    $_SESSION["ultimoAcceso"] = date("Y-n-j H:i:s");
    //defino la fecha y hora de inicio de sesión en formato aaaa-mm-dd hh:mm:ss
    header ("Location: aplicacion.php");
}else {
    //si no existe le mando otra vez a la portada
    header("Location: index.php?errorusuario=si");
}
?>
```

**Cierre de sesión al cerrar el navegador en PHP: módulo de control de datos**  
Verás reflejados los cambios al script anterior, indicados en **negritas**.

```
<?
//si es necesario cambiar la config. del php.ini desde tu script
ini_set("session.use_only_cookies","1");
ini_set("session.use_trans_sid","0");

//iniciamos la sesión
session_name("loginUsuario");
session_start();
session_set_cookie_params(0, "/", $HTTP_SERVER_VARS["HTTP_HOST"], 0);
//cambiamos la duración a la cookie de la sesión

//antes de hacer los cálculos, compruebo que el usuario está logueado
//utilizamos el mismo script que antes
if ($_SESSION["autenticado"] != "SI") {
    //si no está logueado lo envío a la página de autenticación
    header("Location: index.php");
} else {
    //sino, calculamos el tiempo transcurrido
    $fechaGuardada = $_SESSION["ultimoAcceso"];
    $ahora = date("Y-n-j H:i:s");
    $tiempo_transcurrido = (strtotime($ahora)-strtotime($fechaGuardada));

    //comparamos el tiempo transcurrido
    if($tiempo_transcurrido >= 600) {
        //si pasaron 10 minutos o más
        session_destroy(); // destruyo la sesión
        header("Location: index.php"); //envío al usuario a la pag. de autenticación
        //sino, actualizo la fecha de la sesión
    }else {
        $_SESSION["ultimoAcceso"] = $ahora;
    }
}
?>
```

*Artículo por **Eugenia Bahit***

## Autenticación PHP para múltiples usuarios usando MySQL

Vamos a ver las páginas PHP que necesitaríamos para realizar un acceso restringido por clave y contraseña para múltiples usuarios, donde cada uno tenga unos datos de acceso propios.

**Nota:** Este artículo viene a complementar el manual [Sistema Autenticación en PHP](#). De hecho, en este artículo sólo vamos a tratar la página que recoge los datos del usuario (su nombre y contraseña) y comprueba si son correctos, redireccionando a la aplicación segura (si los datos se corresponden con algún usuario de la base de datos), o a la página de entrada (si los datos no correspondían con ningún usuario registrado).

Lo primero es recordar el esquema de páginas del sistema de autenticación propuesto. Lo podemos ver en el artículo [Funcionamiento del sistema de autenticación en PHP](#). Nosotros vamos a tratar de colocar aquí un código para la página "compruebo datos"

### La base de datos

La base de datos que vamos a utilizar contendrá una tabla para los usuarios, donde cada uno dispondrá, al menos, de dos campos: un nombre de usuario y una contraseña, los dos de tipo texto.

| Tabla usuario    |                |
|------------------|----------------|
| Nombre del campo | Tipo del campo |
| nombre_usuario   | Texto          |
| clave_usuario    | Texto          |

En una base de datos de usuarios, el nombre de usuario debería ser un valor único, irrepetible para otro usuario, es decir, no podremos tener dos usuarios con el mismo nombre. Por esta razón, el campo nombre\_usuario podría ser la clave principal de la tabla, aunque también podríamos haber creado un campo adicional, llamado por ejemplo id\_usuario, de tipo autonumérico y colocarlo como clave principal.

Para conseguir no insertar dos usuarios con el mismo nombre de usuario, a la hora de insertarlos en la tabla, comprobaremos que no haya ningún usuario ya introducido con el nombre de usuario que se pretende insertar. Este paso, aunque importante, no lo vamos a ver, pues sólo nos vamos a centrar en decidir si un usuario puede entrar o no en la aplicación, suponiendo que los usuarios se encuentran ya insertados en la base de datos.

En el ejemplo suponemos que utilizamos una base de datos MySQL, sin embargo, cualquier tipo de base de datos podrá servir para unos objetivos como los que nos proponemos.

### El funcionamiento del script

El script que se utilizará para decidir si un usuario puede o no entrar en la aplicación es muy sencillo. Simplemente hace una llamada a la base de datos para comprobar si los datos de autenticación escritos por el visitante (usuario y contraseña) corresponden con los de algún usuario. En caso de que así sea, se permite la entrada y de no ser así, se deniega.

**Nota:** Este script fue comentado en una versión simplificada en el artículo [Control de los datos de](#)

[autenticación en PHP](#), englobado dentro del manual [Sistema de autenticación PHP](#)

Lo primero sería abrir una conexión con la base de datos y seleccionar la base con la que hemos de trabajar.

```
//conecto con la base de datos
$conn = mysql_connect("servidor","usuario","password");
//selecciono la BBDD
mysql_select_db("nombre_bbdd",$conn);
```

Un segundo paso es construir una sentencia SQL que nos permita comprobar si existe o no un usuario con los datos de autenticación introducidos. Utilizamos una simple sentencia SELECT, sobre la tabla de usuarios, donde se extraen usuarios que tengan el mismo nombre de usuario y la contraseña introducidos en la página de acceso.

```
//Sentencia SQL para buscar un usuario con esos datos
$sql = "SELECT * FROM usuario WHERE nombre_usuario='$usuario' and clave_usuario='$contrasena'";

//Ejecuto la sentencia
$rs = mysql_query($sql,$conn);
```

Si esa sentencia SELECT responde con algún registro encontrado, sabremos que existe un usuario donde sus datos de autenticación corresponden perfectamente con los introducidos. En ese caso podremos realizar las acciones encaminadas a permitir el acceso. Por el contrario, si la sentencia SELECT no encuentra ningún registro, sabremos que no existe un usuario con los datos de autenticación introducidos y por lo tanto, deberemos realizar las acciones encaminadas a restringir el acceso.

```
if (mysql_num_rows($rs)!=0){
    //usuario y contraseña válidos
    //defino una sesion y guardo datos
    session_start();
    session_register("autenticado");
    $autenticado = "SI";
    header ("Location: aplicacion.php");
}else {
    //si no existe le mando otra vez a la portada
    header("Location: index.php?errorusuario=si");
}
```

Las acciones para restringir o permitir el acceso son exactamente iguales a las que veníamos utilizando en el script de control sin utilizar la base de datos. Así que no vamos a comentarlas más, sino que os referimos al artículo donde las explicamos.

El código completo del ejemplo sería el siguiente.

```
<?
//conecto con la base de datos
$conn = mysql_connect("servidor","usuario","password");
//selecciono la BBDD
mysql_select_db("nombre_bbdd",$conn);

//Sentencia SQL para buscar un usuario con esos datos
$sql = "SELECT * FROM usuario WHERE nombre_usuario='$usuario' and clave_usuario='$contrasena'";

//Ejecuto la sentencia
$rs = mysql_query($sql,$conn);

//vemos si el usuario y contraseña es válido
```

```
//si la ejecución de la sentencia SQL nos da algún resultado
//es que si que existe esa combinación usuario/contraseña
if (mysql_num_rows($rs)!=0){
    //usuario y contraseña válidos
    //defino una sesion y guardo datos
    session_start();
    session_register("autenticado");
    $autenticado = "SI";
    header ("Location: aplicacion.php");
}else {
    //si no existe le mando otra vez a la portada
    header("Location: index.php?errorusuario=si");
}
mysql_free_result($rs);
mysql_close($conn);
?>
```

**Nota:** Es importante destacar que esta página no debería contener ningún tipo de texto antes de la apertura de código PHP, ni tan siquiera saltos de línea. Esto es debido a que al final se realiza una redirección y este tipo de instrucciones solamente se puede ejecutar si no se ha escrito todavía ningún carácter en el cuerpo. Para ser más específicos, este es el error que obtenemos si escribimos antes en la página de enviar las cabeceras:

*Warning: Cannot add header information - headers already sent by (output started at /htdocs/ejemplos/autentif-php\_bbdd/control.php:2) in /htdocs/ejemplos/autentif-php\_bbdd/control.php on line 26*

## Otra posibilidad de codificación

Previamente a la publicación de este artículo, Hector Varón, un lector de DesarrolloWeb.com, nos mandó un código de control que es básicamente el mismo que hemos explicado nosotros aunque con algunas diferencias que lo hacen interesante para su publicación.

Lo incluimos para que los interesados lo puedan descargar, y desde aquí mandamos nuestros agradecimientos a Hector.

[Descargar otra posibilidad de código.](#)

*Artículo por **Miguel Angel Alvarez***

## Autenticar usuario y guardar en una cookie con PHP

Vamos a crear un sistema para autenticar usuarios con PHP, con la particularidad que este sistema va a ofrecer al visitante la opción de guardar su usuario, para que la página lo recuerde en sucesivos accesos y no tenga que volver a autenticarse. El usuario se guardará en una cookie para que el navegador pueda recordarlo en sus distintas visitas.

Esta es una opción muy útil para que el visitante no tenga que estar todo el tiempo autenticándose, con su usuario y contraseña, cada vez que accede a la página web. Seguro que es una opción que habremos visto en un montón de sitios web.

Este artículo hace continuación de una serie de talleres y ejemplos que hemos visto anteriormente en el [manual de Autenticación de usuarios en PHP](#). En este taller no vamos a realizar una autenticación muy elaborada, sino una muy simple, para facilitar el desarrollo. Luego la complejidad la aportará la parte de guardar el usuario en una cookie, que no es difícil

de hacer, pero sí requiere de nuevos conocimientos que aplicar.

## Explicación de almacenar el usuario en una cookie

Primero vamos a explicar con palabras el modelo de trabajo que vamos a aplicar para almacenar el usuario en una cookie. No he investigado cómo lo podrán hacer esto otras personas en otros desarrollos, pero creo que he ingeniado una forma adecuada para hacerlo.

Digo esto porque lo primero que se me ocurrió fue meter el nombre de usuario y la clave en unas cookies. Sería simplemente crear un par de cookies en el sistema del usuario con esas dos variables. Pero luego pensando, no me parecía muy atractiva la posibilidad de incluir esa información sensible en unas cookies en el ordenador del usuario, por su hubiera alguna persona que pudiera leerlas, copiarlas y utilizarlas en otro ordenador. Igual no me debería preocupar por ello, pero en cualquier caso no me gustaba la posibilidad de almacenar el nombre de usuario y contraseña, sino almacenar otro tipo de información menos crítica.

Entonces lo que se me ocurrió es almacenar el identificador del usuario. Pero claro, si alguien conseguía crear una cookie en el sistema con cualquier identificador de usuario, podría acceder a la cuenta de ese usuario. Así que había que aplicar otra estrategia adicional para asegurar que ese identificador de usuario no se pueda reproducir. Finalmente, decidí generar un número aleatorio cuando el usuario se conecta a la página y se autentica correctamente, y almacenarlo en dos sitios, primero en el registro del usuario en la base de datos y luego en la cookie.

Así, cuando el visitante decide que quiere que el sitio web le recuerde su cuenta de usuario, para no tener que volver a autenticarse en siguientes accesos, se guardan dos cosas en las cookies del navegador: su identificador de usuario y una marca aleatoria (dicha marca también se almacena en la base de datos, asociada a su usuario). En siguientes accesos primero se comprueba si existen esas cookies en el navegador del visitante y si el conjunto de identificador de usuario y la marca aleatoria almacenados en la cookie coincide con lo que tenemos en la base de datos.

Dicho de otra manera, en la cookie guardamos el identificador de usuario. Además, generamos un número aleatorio que guardamos en dos sitios: 1) en la tabla de usuario, en el registro correspondiente al usuario que desea que se le recuerde la clave y 2) en una cookie. Luego cuando el usuario se conecta de nuevo, no sólo se comprueba que tenga la cookie con el identificador de usuario, sino que tenga la otra cookie con la marca aleatoria y que sea la misma que tenemos en la base de datos para ese mismo usuario.





Espero que el sistema se pueda entender. No obstante, espero clarificarlo aun más a medida que explique el código PHP y la base de datos que vamos a utilizar.

## Tabla de usuario

En la tabla de usuario que hemos creado para este ejemplo tenemos 4 campos:

- Identificador de usuario
- Nombre de usuario
- Clave de acceso
- Marca aleatoria que se ha metido en la cookie

Para el ejemplo hemos creado la tabla e insertado un par de usuarios para hacer pruebas. Tendrá una forma como esta:

| ←T→  | id_usuario | usuario | clave | cookie |
|--|------------|---------|-------|--------|
| <input type="checkbox"/>   | 1          | pepe    | 1234  |        |
| <input type="checkbox"/>   | 2          | juan    | 1111  |        |

## Código de la página PHP

Ahora voy a explicar por partes el código PHP para autenticar al usuario y guardar la información de acceso en la cookie, así como la parte de comprobar si el usuario tenía la cookie con su acceso guardado en el ordenador.

Primero vamos a empezar mostrando el formulario HTML:

```
<form action="prueba-cookies.php" method="post">
Usuario: <input type="text" name="usuario">
<br>
Clave: <input type="text" name="clave">
<br>
<input type="checkbox" name="guardar_clave" value="1"> Memorizar el usuario en este ordenador
<br>
<input type="submit" value="Entrar">
</form>
```

Como vemos, tiene los campos para escribir el nombre de usuario y la clave y un campo checkbox adicional para que el usuario marque si quiere que su acceso se guarde en su ordenador.

Ahora voy a mostrar el código que utilizaríamos para recibir por post, del formulario de autenticación, el nombre de usuario y contraseña. Este código también tiene que detectar si el usuario quería que se guardase su acceso en el ordenador.

```
//debería comprobar si el usuario es correcto
$sql = "select * from usuario where usuario = '" . $_POST["usuario"] . "' and clave='" . $_POST["clave"] . "'";
//echo $sql;
$rs = mysql_query($sql);
if (mysql_num_rows($rs)==1){
    //TODO CORRECTO!! He detectado un usuario
    $usuario_encontrado = mysql_fetch_object($rs);
    //ahora debo de ver si el usuario quería memorizar su cuenta en este ordenador
    if ($_POST["guardar_clave"]=="1"){
        //es que pidió memorizar el usuario
        //1) creo una marca aleatoria en el registro de este usuario
        //alimentamos el generador de aleatorios
        mt_srand (time());
        //generamos un número aleatorio
        $numero_aleatorio = mt_rand(1000000,999999999);
        //2) meto la marca aleatoria en la tabla de usuario
        $sql = "update usuario set cookie='$numero_aleatorio' where id_usuario=" . $usuario_encontrado->id_usuario;
        mysql_query($sql);
        //3) ahora meto una cookie en el ordenador del usuario con el identificador del usuario y la cookie aleatoria
        setcookie("id_usuario_dw", $usuario_encontrado->id_usuario, time()+(60*60*24*365));
        setcookie("marca_aleatoria_usuario_dw", $numero_aleatorio, time()+(60*60*24*365));
    }
    echo "Autenticado correctamente";
    //header ("Location: contenidos_protegidos_cookie.php");
}
else{
    echo "Fallo de autenticación!";
    echo "<p><a href='prueba-cookies.php'>Volver</a>";
}
```



```
}
```

Para comprobar si los datos de autenticación que recibimos por el formulario son correctos, hacemos una sentencia SQL. La ejecutamos y si nos da como resultado que tenemos un registro encontrado en la tabla de usuarios, es que el nombre de usuario y clave corresponden con el de algún usuario.

Luego con la línea

```
if ($_POST["guardar_clave"]=="1"){
```

Comprobamos si el visitante había pedido que se almacenase la clave en su ordenador. Entonces hay que generar las cookies correspondientes, que habíamos comentado anteriormente en este artículo.

Lo hacemos en tres pasos:

- Genero un número aleatorio para que nos sirva de marca.
- Inserto la marca aleatoria en la tabla de usuarios, haciendo un update en el registro del usuario autenticado que habíamos detectado anteriormente.
- Genero y coloco en el ordenador del usuario las dos cookies para guardar su acceso en el navegador: el identificador del usuario y la marca aleatoria. Hemos creado las cookies para que se almacenen durante un año en el ordenador del usuario.

**Nota:** Es importante señalar que, para colocar o crear cookies en el navegador del visitante, debemos hacerlo antes de que se hayan enviado las cabeceras de http, es decir, antes de haber escrito cualquier texto en la página. Si no, nos podría dar un error de http headers already sent.

Por último veamos el código PHP para ver si detectamos las cookies en el navegador de un usuario autenticado anteriormente en el sistema y guardado en el ordenador del usuario.

```
//primero tengo que ver si el usuario está memorizado en una cookie
if (isset($_COOKIE["id_usuario_dw"]) && isset($_COOKIE["marca_aleatoria_usuario_dw"])){
    //Tengo cookies memorizadas
    //además voy a comprobar que esas variables no estén vacías
    if ($_COOKIE["id_usuario_dw"]!="" || $_COOKIE["marca_aleatoria_usuario_dw"]!=""){
        //Voy a ver si corresponden con algún usuario
        $ssql = "select * from usuario where id_usuario=" . $_COOKIE["id_usuario_dw"] . " and cookie=" .
$_COOKIE["marca_aleatoria_usuario_dw"] . " and cookie<>"";
        $rs = mysql_query($ssql);
        if (mysql_num_rows($rs)==1){
            echo "<b>Tengo un usuario correcto en una cookie</b>";
            $usuario_encontrado = mysql_fetch_object($rs);
            echo "<br>Eres el usuario número " . $usuario_encontrado->id_usuario . ", de nombre " .
$usuario_encontrado->usuario;
            //header ("Location: contenidos_protegidos_cookie.php");
        }
    }
}
```

Como primer paso compruebo si existen las cookies con el identificador del usuario y la mencionada marca aleatoria. Además, hacemos una comprobación adicional para ver si alguna de las dos cookies contiene un string vacío, porque en ese caso no están guardadas correctamente y no nos sirven.

Si todo ha ido bien, miramos en la base de datos si el usuario con identificador determinado en la cookie tiene la marca aleatoria igual que la que tenía la cookie del navegador del visitante.

Además, en la consulta en la base de datos también nos aseguramos que la marca aleatoria sea distinta de "", porque entonces es un usuario que nunca había pedido que se guardasen sus datos en el ordenador.

Si esa consulta daba un registro, es que corresponde con un usuario que se había almacenado en el ordenador y es el usuario que estaba autenticado anteriormente y guardado su acceso.

## Conclusión

Hasta aquí he comentado todo lo que necesitamos saber para crear la infraestructura para que la página web recuerde la clave del usuario y no tenga que autenticarse cada vez que accede al sitio. El código que hemos mostrado podría completarse con una serie de mejoras o personalizaciones para adaptarlo a nuestras necesidades, pero seguro que sirve de guía para el interesado.

Ahora presento el código completo de la página de este ejemplo:

```
<?
//conecto con la base de datos
$conn = mysql_connect("servidor","usuario","clave");
//selecciono la BBDD
mysql_select_db("base de datos",$conn);

//primero tengo que ver si el usuario está memorizado en una cookie
if (isset($_COOKIE["id_usuario_dw"]) && isset($_COOKIE["marca_aleatoria_usuario_dw"])){
    //Tengo cookies memorizadas
    //además voy a comprobar que esas variables no estén vacías
    if ($_COOKIE["id_usuario_dw"]!="" || $_COOKIE["marca_aleatoria_usuario_dw"]!=""){
        //Voy a ver si corresponden con algún usuario
        $ssql = "select * from usuario where id_usuario=" . $_COOKIE["id_usuario_dw"] . " and cookie=" .
        $_COOKIE["marca_aleatoria_usuario_dw"] . " and cookie<>"";
        $rs = mysql_query($ssql);
        if (mysql_num_rows($rs)==1){
            echo "<b>Tengo un usuario correcto en una cookie</b>";
            $usuario_encontrado = mysql_fetch_object($rs);
            echo "<br>Eres el usuario número " . $usuario_encontrado->id_usuario . ", de nombre " .
            $usuario_encontrado->usuario;
            //header ("Location: contenidos_protegidos_cookie.php");
        }
    }
}

if ($_POST){
    //es que estamos recibiendo datos por el formulario de autenticación (recibo de $_POST)

    //debería comprobar si el usuario es correcto
    $ssql = "select * from usuario where usuario = " . $_POST["usuario"] . " and clave=" . $_POST["clave"] . """;
    //echo $ssql;
    $rs = mysql_query($ssql);
    if (mysql_num_rows($rs)==1){
        //TODO CORRECTO!! He detectado un usuario
        $usuario_encontrado = mysql_fetch_object($rs);
        //ahora debo de ver si el usuario quería memorizar su cuenta en este ordenador
        if ($_POST["guardar_clave"]=="1"){
            //es que pidió memorizar el usuario
            //1) creo una marca aleatoria en el registro de este usuario
            //alimentamos el generador de aleatorios
            mt_srand (time());
            //generamos un número aleatorio
            $numero_aleatorio = mt_rand(1000000,999999999);
            //2) meto la marca aleatoria en la tabla de usuario
        }
    }
}
```

```
$ssql = "update usuario set cookie='$numero_aleatorio' where id_usuario=" . $usuario_encontrado-
>id_usuario;
mysql_query($ssql);
//3) ahora meto una cookie en el ordenador del usuario con el identificador del usuario y la cookie aleatoria
setcookie("id_usuario_dw", $usuario_encontrado->id_usuario , time()+(60*60*24*365));
setcookie("marca_aleatoria_usuario_dw", $numero_aleatorio, time()+(60*60*24*365));
}
echo "Autenticado correctamente";
//header ("Location: contenidos_protegidos_cookie.php");

}else{
    echo "Fallo de autenticación!";
    echo "<p><a href='prueba-cookies.php'>Volver</a>";
}

}else{
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>cookies para detectar usuario</title>
</head>

<body>

<form action="prueba-cookies.php" method="post">
Usuario: <input type="text" name="usuario">
<br>
Clave: <input type="text" name="clave">
<br>
<input type="checkbox" name="guardar_clave" value="1"> Memorizar el usuario en este ordenador
<br>
<input type="submit" value="Entrar">
</form>
<br>
<br>
<b>Usuarios válidos:</b>
<br>
<br>
    User: pepe
    <br>
    Clave: 1234
    <br>
    <br>
    User: juan
    <br>
    Clave: 1111

</body>
</html>

<?
}
?>
```

Por último, dejo aquí el enlace para [ver el ejemplo creado en funcionamiento](#).

*Artículo por **Miguel Angel Alvarez***