

## Algunos Trucos y Consejos de PHP que te pueden ayudar.



- **Desarrolla con el reporting de errores activado**

Ya hablamos de él cuando nos conectamos a las BD, pero cuidado con lo que los *phperos* llaman el "pantallazo blanco de la muerte". Un error 500 que no da ninguna información y que puede resultar tremendamente frustrante. Para evitarlo, sin tener

que modificar el fichero php.ini, durante la fase de desarrollo tan sólo es necesario incluir las dos siguientes líneas al principio del código:

```
error_reporting(E_ALL);
```

```
ini_set('display_errors',1);
```

Esto te permitirá ver tanto los errores fatales que producen el temido pantallazo blanco como warnings o notices que pueden ser bugs a arreglar. Luego, claro está, que no se te olvide eliminarlas al subir tu web a producción.

- **Evita la inyección SQL**

Posiblemente una de las causas de la mala prensa de PHP sean los agujeros de seguridad (Cross-Site Scripting, Cross-Site Request Forgeries...) que se te pueden colar en cuanto no seas un poco meticuloso. De todos ellos el más conocido (y el más sencillo de prevenir) es el de la **SQL Injection**: *"la inserción de código SQL invasor con el fin de alterar el funcionamiento normal del programa y lograr que se ejecute la porción de código invasor en la base de datos"* con funestas consecuencias.



¿Cómo lo evitamos? [Hay varias maneras](#), pero la más sencilla es **escapando siempre cualquier variable** (y no sólo los inputs del usuario) **que vayamos a utilizar contra la base de datos**. Tal que así:

```
$con = mysqli_connect("localhost","user", "pass","db");  
$ciudad="'sHertogenbosch";
```

```
$ciudad=mysqli_real_escape_string($con,$ciudad);
```

```
/*esta consulta con $ciudad escapada funcionará perfectamente, de no hacerlo,
nos daría problemas con la comilla simple*/

if (mysqli_query($con,"INSERT into miCiudad (Nombre) VALUES '$ciudad'")) {
    printf("%d fila insertada.\n", mysqli_affected_rows($con));
}

mysqli_close($con);
```

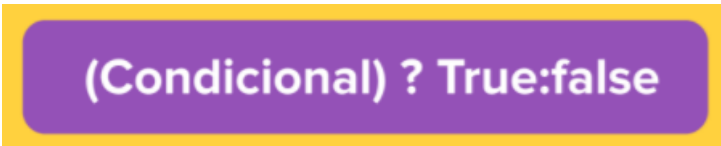
**Claro que nosotros vamos a utilizar PDO y consultas preparadas, que ya te garantizan esta seguridad.**

- **Usa las funciones `_once()` con precaución**

A la hora de llamar a otros ficheros, librerías o clases podemos optar por la función **`include()`** o la función **`require()`**. La primera da un warning de no encontrar el fichero y sigue la ejecución. La segunda da un error fatal y para la ejecución. Bien, hasta aquí todo claro, PHP de manual.

Sin embargo también tenemos la opción de utilizar las funciones **`include_once()`** y **`require_once()`**, que tienen la misma funcionalidad pero **evitan que los ficheros, clases o librerías cargados puedan cargarse de nuevo** causando duplicidades y estados no deseados en el código. Esto está genial a priori pero lo consiguen a costa de **una disminución de rendimiento bastante notoria**. Por lo tanto **mejor ocúpate tu mismo**, siempre que puedas, **de evitar estas duplicidades** revisando bien el código y todas las llamadas en el mismo.

- **Aprende a manejar los operadores ternarios**



(Condicional) ? True:false

Los operadores ternarios son una buena alternativa a las construcciones IF simples: **en una misma línea tienes el condicional y los resultados del TRUE y el FALSE**. Aquí tenemos un ejemplo:

```
$nombre = (!empty($_GET['nombre'])?$_GET['nombre'] : 'Luis');
```

La variable tendrá el valor del parámetro GET y de no existir o tener valor *false*, tendrá el literal *Luis*.

Y ya que estamos, recuerda el **operador Fusion Null** del que hablamos en los primeros temas. Muy útil en estos casos. Os recuerdo su uso más abajo.

- **Usa un Switch en vez de encadenar ifs a lo loco**

Tened en cuenta que no necesitamos utilizar ifs para todo. Switch es ligeramente más rápido que If, pero dejando de lado el tema rendimiento resulta horrible ver esa cantidad obscena de if-elseif-else encadenados que hacen ilegibles algunos códigos. Usa Switch si puedes, que para eso PHP es un lenguaje moderno y trae esta construcción.

Y si además, **colocas antes los casos que consideres que van a ser más usados**, el rendimiento será todavía mejor.

- **¿Usar comillas simples en vez de dobles?**

Esto puede parecer una perogrullada pero usar comillas simples (") en vez de dobles (") es el doble de rápido, lo parece sugerir la utilización de comillas simples siempre que se pueda. El rendimiento de tu servidor te lo agradecerá.

Vale, ok, eso dicen, pero según vemos en los resultados de *The Php Benchmark*, **apenas hay diferencia entre la eficiencia entre el uso de unas comillas y otras**. Así que podemos quedarnos tranquilos: programando en versiones modernas de PHP, donde se ha mejorado notablemente el rendimiento del parseo de variables dentro de dobles comillas, podemos elegir la que más nos guste o nos deje un **código más legible**, factor más que importante sobretodo si juntamos código PHP con etiquetas HTML, como es nuestro caso.

- **Precálculo del Tamaño de un Array: Sizeof / Count**

Esta es muy fácil: absolutamente sí, **siempre, debemos precalcular el tamaño de un array de PHP a la hora de iterarlo**, utilizando de manera indistinta la función *count* o su alias *sizeof*. Debajo en la imagen podemos ver la gran diferencia entre hacerlo o no, que nos indica la página The Php BenchMark.

```
//sin precálcuLo
for ($i=0; $i<sizeof($array); $i++);

//con precálcuLo
$size= count($array);
for ($i=0; $i<$size; $i++);
```

+ 111 %	With pre calc - count()	Total time: 383 µs
+ 75744 %	Without pre calc - count()	Total time: 261310 µs
+ 100 %	With pre calc - sizeof()	Total time: 345 µs
+ 52677 %	Without pre calc - sizeof()	Total time: 181732 µs

## ● **PHP encripta tus passwords por ti**



A partir de **PHP 5.5**, nos proporciona **encriptación nativa** para los passwords que quieras almacenar en base de datos de una forma tan sencilla como:

```
$pass_enc = password_hash($pass_user, PASSWORD_DEFAULT);
```

¿Y chequear si la password es correcta? Igual de sencillo:

```
if(password_verify($pass_user, $pass_bd)){
    //usuario correctamente identificado
}
```

Adiós engorros y calentamientos de cabeza con sha1() o md5().

## ● **OPERADOR FUSION NULL**

Acordaos del operador fusión null que hemos visto. Muy útil...

### Operador de fusión de null

El operador de fusión de null (??) se ha añadido como aliciente sintáctico para el caso común de la necesidad de utilizar un operador ternario junto con [isset\(\)](#). Devuelve su primer operando si existe y no es **NULL**; de lo contrario devuelve su segundo operando.

```
<?php
// Obtener el valor de $_GET['usuario'] y devolver 'nadie'
// si no existe.
$nombre_usuario = $_GET['usuario'] ?? 'nadie';
// Esto equivale a:
$nombre_usuario = isset($_GET['usuario']) ? $_GET['usuario'] : 'nadie';

// La fusión se puede encadenar: esto devolverá el primer
// valor definido de $_GET['usuario'], $_POST['usuario'],
// y 'nadie'.
$nombre_usuario = $_GET['usuario'] ?? $_POST['usuario'] ?? 'nadie';
?>
```

- **Pasa variables por referencia**

Lo de las variables por referencia no es sólo para Java y demás lenguajes compilados, en PHP también **puedes pasar los parámetros por referencia a una función y así esta podrá actualizar su valor** sin necesidad de devolver nada o declarar engorrosas y olvidadizas variables globales. Tan fácil como anteponer un ampersand (&) y listo:

```
function cuadrado(&number) {  
    $number *= $number;  
}  
  
$number = 2;  
square($number);  
echo $number; //devuelve 4
```

Vale, bien, **pero no abuséis de esto**. Puede dar lugar a **mucha confusión a la hora de programar y depurar**. **Mientras sea posible, pasad variables a las funciones, y si modifican algo, que lo devuelvan al acabar su tarea**. Muchos más claro todo si trabajamos así.

---

## BIBLIOGRAFÍA

<http://www.phptherightway.com/> Una página muy útil y actualizada de la forma correcta de utilizar un montón de recursos en php

<https://www.genbetadev.com/php/13-trucos-y-consejos-de-php-que-pueden-hacerte-la-vida-profesional-mas-facil>