

# PHP 7

## Cambios y novedades

# PHP 7

## Cambios retroincompatibles

# Cambios retrocompatibles

## Antigua y nueva evaluación de expresiones indirectas

El acceso indirecto a variables, propiedades y métodos ahora se evalúa en PHP7 estrictamente de izquierda a derecha

Expresión	Interpretación de PHP 5	Interpretación de PHP 7
<code>\$\$foo[ 'bar' ][ 'baz' ]</code>	<code>\${\$\$foo[ 'bar' ][ 'baz' ]}</code>	<code>( \$\$foo )[ 'bar' ][ 'baz' ]</code>
<code>\$foo-&gt;\$bar[ 'baz' ]</code>	<code>\$foo-&gt;{\$bar[ 'baz' ]}</code>	<code>( \$foo-&gt;\$bar )[ 'baz' ]</code>
<code>\$foo-&gt;\$bar[ 'baz' ]()</code>	<code>\$foo-&gt;{\$bar[ 'baz' ]}()</code>	<code>( \$foo-&gt;\$bar )[ 'baz' ]()</code>
<code>Foo::\$bar[ 'baz' ]()</code>	<code>Foo::{\$bar[ 'baz' ]}()</code>	<code>( Foo::\$bar )[ 'baz' ]()</code>

# Cambios retrocompatibles

## list() ya no asigna variables en orden inverso

```
list($a[], $a[], $a[]) = [1, 2, 3];
var_dump($a);
```

Resultado con PHP5

```
array(3) {
    [0]=>
    int(3)
    [1]=>
    int(2)
    [2]=>
    int(1)
}
```

Resultado con PHP7

```
array(3) {
    [0]=>
    int(1)
    [1]=>
    int(2)
    [2]=>
    int(3)
}
```

# Cambios retrocompatibles

## Se han eliminado las asignaciones vacías de list()

Las siguientes instrucciones ya no son válidas:

```
list() = $a;  
list(,,) = $a;  
list($x, list(), $y) = $a;
```

# Cambios retrocompatibles

## global solamente acepta variables simples

Ya no se pueden utilizar *variables variables* con la palabra reservada *global*. Se puede emplear la sintaxis de llaves para emular el comportamiento anterior si fuera necesario:

```
function f() {  
    // Válido solamente en PHP 5.  
    global $$foo;  
  
    // Válido en PHP 5 y 7.  
    global ${$foo};  
}
```

# Cambios retrocompatibles

## La iteración de `foreach` por valor opera sobre una copia del array

Esto significa que los cambios realizados a un array durante una iteración no afectarán a los valores que se recorren.

## Se ha mejorado el comportamiento de la iteración de `foreach` por referencia

Recorriendo por referencia, se realiza un seguimiento de los cambios hechos en el array. Por ejemplo, al añadir valores a un array mientras se recorre dará como resultado también el recorrido de los valores añadidos:

```
$array = [0];
foreach ($array as &$val) {
    var_dump($val);
    $array[1] = 1;
}
```

Salida con PHP 5: int(0)

Salida con PHP 7: int(0)  
int(1)

# Cambios retrocompatibles

**foreach ya no cambia el puntero de arrays interno**

```
$array = [0, 1, 2];
foreach ($array as &$val) {
    var_dump(current($array));
}
```

Resultado con PHP5

```
int(1)
int(2)
bool(false)
```

Resultado con PHP7

```
int(0)
int(0)
int(0)
```

# Cambios retrocompatibles

## Un string hexadecimal ya no se considera numérico

```
var_dump("0x123" == "291");
var_dump(is_numeric("0x123"));
var_dump("0xe" + "0x1");
var_dump(substr("foo", "0x1"));
```

Resultado con PHP5

```
bool(true)
bool(true)
int(15)
string(2) "oo"
```

Resultado con PHP7

```
bool(false)
bool(false)
int(0)
```

```
Notice: A non well formed numeric value
encountered in /tmp/test.php on line 5
string(3) "foo"
```

# Cambios retrocompatibles

## Funciones eliminadas

### **call\_user\_method()** y **call\_user\_method\_array()**

Estas funciones estaban obsoletas en PHP 4.1.0 en favor de **call\_user\_func()** y **call\_user\_func\_array()**.

### **Todas las funciones ereg\***

Se han eliminado todas las funciones de ereg. Se recomienda **PCRE** como alternativa.

### **Alias de mcrypt**

Se ha eliminado la función obsoleta **mcrypt\_generic\_end()** en favor de **mcrypt\_generic\_deinit()**.

También se han eliminado las funciones obsoletas **mcrypt\_ecb()**, **mcrypt\_cbc()**, **mcrypt\_cfb()** y **mcrypt\_ofb()** en favor del uso de **mcrypt\_decrypt()** con la constante **MCRYPT\_MODE\_\*** apropiada.

### **Todas las funciones de ext/mysql**

Se han eliminado todas las funciones de ext/mysql. Utilizar **ext/mysqli** o **PDO\_MySQL**.

### **Todas las funciones ext/mssql**

Se han eliminado todas las funciones de ext/mssql. Las alternativas son: **PDO\_SQLSRV**, **PDO\_ODBC**, **SQLSRV** o la **API de ODBC** unificada.

# Cambios retrocompatibles

## Funciones eliminadas

### Alias de `intl`

Se han eliminado los alias obsoletos `datefmt_set_timezone_id()` y `IntlDateFormatter::setTimeZoneID()` en favor de `datefmt_set_timezone()` y `IntlDateFormatter::setTimeZone()`, respectivamente.

### `set_magic_quotes_runtime()`

Se ha eliminado `set_magic_quotes_runtime()`, junto con su alias `magic_quotes_runtime()`. Estaban obsoletos en PHP 5.3.0, y de hecho se quedaron sin funcionalidad con la eliminación de las comillas mágicas en PHP 5.4.0.

### `set_socket_blocking()`

Se ha eliminado el alias obsoleto `set_socket_blocking()` en favor de `stream_set_blocking()`.

# Cambios retrocompatibles

## Funciones eliminadas

### **dl() en PHP-FPM**

dl() ya no se puede utilizar en PHP-FPM. Permanece funcional en CLI y en SAPI embebidas.

### **Funciones de Type1 de GD**

Se ha eliminado el soporte para fuentes Type1 de PostScript de la extensión GD, resultado en la eliminación de las siguientes funciones:

**imagepsbbox()**

**imagepsencodefont()**

**imagepsextendfont()**

**imagepsfreefont()**

**imagepsloadfont()**

**imagepsslantfont()**

**imagepstext()**

Se recomienda utilizar fuentes TrueType y sus asociadas en su lugar.

# Cambios retrocompatibles

## DirectivasINI eliminadas

Las siguientes directivasINI han sido eliminadas, así como sus características asociadas:

- **always\_populate\_raw\_post\_data**
- **asp\_tags**
- **xsl.security\_prefs**

En lugar de la directiva **xsl.security\_prefs**, debería llamarse al método **XsltProcessor::setSecurityPrefs()** para controlar las preferencias de seguridad en función de cada procesador.

# Cambios retrocompatibles

## Eliminación de las etiquetas ASP y de script de PHP

Se ha eliminado el soporte para usar etiquetas ASP y de script para delimitar código de PHP. Las etiquetas afectadas son:

Etiqueta de apertura	Etiqueta de cierre
<%	%>
<%=	%>
<script language="php">	</script>

# Cambios retrocompatibles

## yield ahora es un operador asociativo derecho

El constructor yield ya no requiere paréntesis, ya que ha pasado a ser un operador asociativo derecho con precedencia entre print y =>.

Esto puede resultar en un cambio de comportamiento:

```
echo yield -1;  
// Anteriormente era interpretado como  
echo (yield) - 1;  
// Y ahora es interpretado como  
echo yield (-1);  
  
yield $foo or die;  
// Anteriormente era interpretado como  
yield ($foo or die);  
// Y ahora es interpretado como  
(yield $foo) or die;
```

# Cambios retrocompatibles

## Las funciones no pueden tener varios parámetros con el mismo nombre

Ya no es posible definir dos o más parámetros de función con el mismo nombre. Por ejemplo, el siguiente método generará un error **E\_COMPILE\_ERROR**:

```
function foo($a, $b, $c, $c) {  
    //  
}
```

# Cambios retrocompatibles

## Las funciones que inspeccionan argumentos informan del valor del parámetro actual

**func\_get\_arg()**, **func\_get\_args()**, **debug\_backtrace()** y las excepciones de información de rastreo ya no informarán del valor original que fue pasado a un parámetro, en su lugar proporcionarán el **valor actual** (el cual podría haber sido modificado).

```
function foo($x) {  
    $x++;  
    var_dump(func_get_arg(0));  
}  
foo(1);
```

Salida con PHP 5: int(1)

Salida con PHP 7: int(2)

# Cambios retrocompatibles

## Las sentencias **switch** no pueden tener varios bloques **default**

Ya no es posible definir dos o más bloques **default** en una sentencia **switch**. Por ejemplo, la siguiente sentencia switch generará un error **E\_COMPILE\_ERROR**:

```
switch (1) {  
    default:  
        break;  
    default:  
        break;  
}
```

# Cambios retrocompatibles

## Eliminación de \$HTTP\_RAW\_POST\_DATA

\$HTTP\_RAW\_POST\_DATA ya no está disponible.

Debería usarse el flujo **php://input** en su lugar.

## Eliminación de los comentarios # en ficherosINI

Se ha eliminado el soporte para los comentarios prefijados con # en los ficherosINI.

Se debe emplear en su lugar ; (punto y coma).

Este cambio se aplica a **php.ini**, así como a ficheros manejados por **parse\_ini\_file()** y **parse\_ini\_string()**.

# Cambios retrocompatibles

## La extensión JSON se ha reemplazado por JSOND

Se ha reemplazado la extensión JSON por JSOND ocasionando tres roturas de RC (retocompatibilidad):

- La primera, un número no puede finalizar con un punto decimal (esto es, 34. debe cambiarse para que sea o 34.0 o 34).
- La segunda, al utilizar notación científica, el exponente e no debe seguir inmediatamente a un punto decimal (esto es, 3.e3 se debe cambiar para que sea o 3.0e3 o 3e3).
- Finalmente, una cadena vacía ya no se considera un JSON válido.

# Cambios retrocompatibles

## Extensiones y SAPI eliminadas

Extensiones eliminadas:

- ereg
- mssql
- mysql
- sybase\_ct

SAPI eliminadas:

- aolserver
- apache
- apache\_hooks
- apache2filter
- caudium
- continuity
- isapi
- milter
- nsapi
- phttpd
- pi3web
- roxen
- thttpd
- tux
- webjames

# PHP 7

## Características obsoletas

# Características obsoletas

## Constructores al estilo de PHP4

Los constructores al estilo de PHP4 (métodos que tienen el mismo nombre que la clase donde están definidos) están obsoletos y serán eliminados en el futuro.

PHP 7 emitirá un error **E\_DEPRECATED** si un constructor de PHP 4 es el **único** definido dentro de una clase. Las clases que implementen un método **`__construct()`** no se ven afectadas.

```
class foo {
    function foo() {
        echo 'Soy el constructor';
    }
}
```

# Características obsoletas

## Llamadas estáticas a métodos no estáticos

Una llamada estática a un método que no esté declarado como *static* está obsoleta y podría ser eliminada en el futuro.

```
class foo {  
    function bar() {  
        echo '¡No soy estático!';  
    }  
}  
  
foo::bar();
```

# Características obsoletas

## Opción 'salt' de password\_hash()

La opción 'salt' de la función **password\_hash()** está obsoleta para prevenir a los desarrolladores de generar sus propias salts (usualmente inseguras). La función en sí genera una salt criptográficamente segura cuando el desarrollador no proporciona ninguna. Por tanto, no debería ser necesaria la generación de sales personalizadas.

## Opción de contexto SSL capture\_session\_meta

La opción de contexto SSL capture\_session\_meta ha sido declarada obsoleta. Los metadatos de SSL ahora están disponibles a través de la función stream\_get\_meta\_data().

## Función ldap\_sort()

La función ldap\_sort() está obsoleta:

## **ext/mcrypt**

Obsoleta en PHP 7.1 en favor de **OpenSSL**, y eliminada del núcleo en PHP 7.2.

# Características obsoletas

## Cadenas de textos sin entrecomillar

Antes emitían un **E\_NOTICE**, en PHP 7.2 emiten un **E\_WARNING**. En la siguiente “major version” de PHP provocarán un error.

## **png2wbmp()** and **jpeg2wbmp()**

### **autoload()**

Se debe utilizar **spl\_autoload\_register()**

## **create\_function()**

La alternativa es utilizar funciones anónimas

## **parse\_str() sin un segundo parámetro**

Por razones de seguridad se debe utilizar el segundo parámetro.

## **gmp\_random()**

Las alternativas son **gmp\_random\_bits()** y **gmp\_random\_range()**.

# Características obsoletas

## each()

Utilizar foreach()

## assert() con un argumento string

Utilizar expresiones boleadas.

## Argumento \$errcontext en los handlers de errores

La alternativa es utilizar algún debugger.

## read\_exif\_data()

La alternativa es exif\_read\_data().

# PHP 7

## Novedades

# Novedades

## Declaraciones de tipo escalar:

- int
- string
- float
- bool

```
function sumaDeEnteros(int ...$enteros)
{
    return array_sum($enteros);
}

var_dump(sumaDeEnteros(2, '3', 4.1));
```

# Novedades

## Declaraciones de tipo escalar:

- int
- string
- float
- bool

```
<?php
declare(strict_types=1);

function sumaDeEnteros(int ...$enteros)
{
    return array_sum($enteros);
}

var_dump(sumaDeEnteros(2, '3', 4.1));
```

# Novedades

## Declaraciones del tipo de devolución

```
function sumarArrays(array ...$arrays): array
{
    return array_map(function(array $array): int {
        return array_sum($array);
    }, $arrays);
}

print_r(sumarArrays([1,2,3], [4,5,6], [7,8,9]));
```

# Novedades

## Tipos nullable

Declarar un argumento como nullable no lo hace opcional. Sigue siendo obligatorio aunque admite que se le pase *null*.

```
function sumarArrays(?array ...$arrays): ?array
{
    return array_map(function(array $array): int {
        return array_sum($array);
    }, $arrays);
}

print_r(sumarArrays([1,2,3], [4,5,6], [7,8,9]));
```

# Novedades

## Retorno de función void

Las funciones declaradas con 'void' como su tipo de retorno deben o bien omitir su sentencia 'return' totalmente, o utilizar una sentencia 'return' vacía. NULL no es un valor de retorno válido para una función 'void'.

```
function intercambiar(&$izquierdo, &$derecho) : void
{
    if ($izquierdo === $derecho) {
        return;
    }

    $tmp = $izquierdo;
    $izquierdo = $derecho;
    $derecho = $tmp;
}
```

# Novedades

## Tipo object (php 7.2)

```
function test(object $obj) : object
{
    // ...
}
```

# Novedades

## Pseudotipo iterable

```
function iterator(iterable $iter)
{
    foreach ($iter as $val) {
        //
    }
}
```

# Novedades

## Operador fusión: ??

```
// Esto:  
$nombre_usuario = $_GET['usuario'] ?? 'nadie';  
// equivale a:  
$nombre_usuario = isset($_GET['usuario']) ? $_GET['usuario'] : 'nadie';  
  
// La fusión se puede encadenar:  
$nombre_usuario = $_GET['usuario'] ?? $_POST['usuario'] ?? 'nadie';
```

# Novedades

## Operador nave espacial: <=>

```
switch ($var1 <=> $var2) {  
    case 0:  
        echo '$var1 y $var2 son iguales';  
    case -1:  
        echo '$var1 es mayor';  
    case 1:  
        echo '$var2 es mayor';  
}
```

# Novedades

## Definir constantes de tipo array con define()

Hasta ahora solamente se podían definir constantes de tipo array con **const**.

```
define( 'ANIMALES' , [  
    'perro',  
    'gato',  
    'pajaro'  
]);
```

# Novedades

## Desestructuración de arrays

```
$datos = [
    [1, 'Tom'],
    [2, 'Fred'],
];

// Estilo de list()
list($id1, $nombre1) = $datos[0];

// Estilo de []
[$id1, $list] = $datos[0];

// Estilo de list()
foreach ($datos as list($id, $nombre)) {
    // aquí va la lógica con $id y $nombre
}

// Estilo de []
foreach ($datos as [$id, $nombre]) {
    // aquí va la lógica con $id y $nombre
}
```

# Novedades

## Soporte para claves en list

```
$datos = [  
    ["id" => 1, "nombre" => 'Tom'],  
    ["id" => 2, "nombre" => 'Fred'],  
];  
  
// Estilo de list()  
list("id" => $id1, "nombre" => $nombre1) = $datos[0];  
  
// Estilo de []  
["id" => $id1, "nombre" => $nombre1] = $datos[0];
```

# Novedades

## Soporte para índices negativos en cadenas

```
var_dump("abcdef"[-2]);  
var_dump(strpos("aabbcc", "b", -3));
```

El resultado del código anterior, sería:

```
string(1) "e"  
int(3)
```

# Novedades

## Clases anónimas

Se pueden definir clases anónimas con ***new class***

```
$app = new Application;  
$app->setLogger(new class {  
    public function log(string $msg) {  
        echo $msg;  
    }  
});
```

Muy útiles sobre todo para testeo.

# Novedades

## Visibilidad de constantes de clase

```
class ConstDemo
{
    const PUBLIC_CONST_A = 1;
    public const PUBLIC_CONST_B = 2;
    protected const PROTECTED_CONST = 3;
    private const PRIVATE_CONST = 4;
}
```

# Novedades

## Multicaptura de excepciones

```
try {  
    // algo de código  
} catch (FirstException | SecondException $e) {  
    // manejar la primera y segunda excepción  
}
```

# Novedades

## Sintaxis para unicode

```
echo "\u{aa}";
```

```
echo "\u{9999}";
```

# Novedades

## Filtros para unserialize

```
// convertir todos los objetos a un objeto __PHP_Incomplete_Class
$data = unserialize($foo, ["allowed_classes" => false]);

// convertir todos los objetos a un objeto __PHP_Incomplete_Class
// excepto a los de MiClase y MiClase2
$data = unserialize($foo, ["allowed_classes" => [ "MiClase", "MiClase2" ] ]);

// comportamiento predeterminado (lo mismo que omitir el segundo argumento)
$data = unserialize($foo, ["allowed_classes" => true]);
```

# Novedades

## Declaraciones de `use` en grupo

```
use un\espacioDeNombres\{ClaseA, ClaseB, ClaseC as C};  
use function un\espacioDeNombres\{fn_a, fn_b, fn_c};  
use const un\espacioDeNombres\{ConstA, ConstB, ConstC};
```

# Novedades

**División entera: *intdiv()***

```
intdiv(10, 3)
```

# Novedades

## Opciones de sesión

session\_start() ahora acepta un array de opciones que prevalecen sobre las directivas de configuración de sesiones establecidas en php.ini.

```
session_start([
    'cache_limiter' => 'private',
    'read_and_close' => true,
]);
```

# Novedades

## Funciones de CSPRNG

Se han añadido dos nuevas funciones para generar números enteros y cadenas de caractéres criptográficamente seguros: **random\_bytes()** y **random\_int()**.

## Mejora en list

Anteriormente, no se garantizaba que `list()` operase correctamente con objetos que implementasen **ArrayAccess**. Esto ha sido arreglado.

# PHP 7

Fin