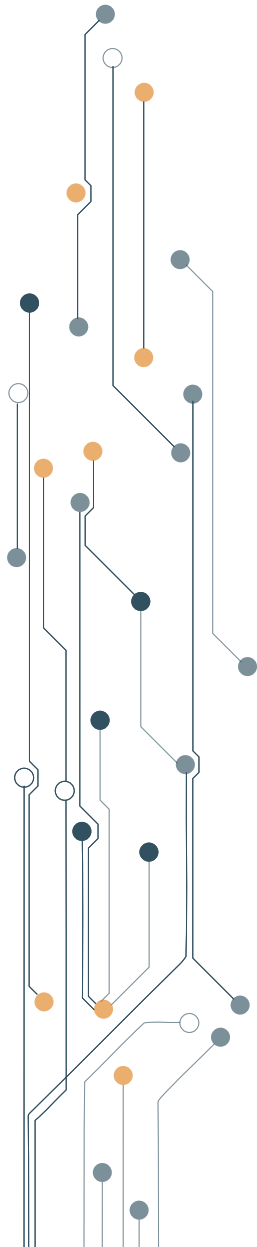




Desarrollo de plugins

Índice



Desarrollo de plugins

1 Cómo funcionan los plugins	3
2 Requisitos	4
2.1 Consideraciones previas	4
2.2 La cabecera del plugin	5
3 Creando un shortcode	6
4 Creando un widget	8

1 Cómo funcionan los plugins

Cuando necesitemos una funcionalidad de la que no dispone el propio WordPress el primer paso es buscar un plugin de calidad que nos solucione el problema. Hemos visto que WordPress cuenta con un gran número de plugins para casi cualquier cosa, pero a veces no son suficientes.

A veces necesitamos una funcionalidad que no implemente aún ningún plugin o al menos no como quisiéramos. Es entonces cuando podemos aprovechar nuestro conocimiento como programadores para lograr nuestro propósito. Hasta ahora hemos visto como hacerlo desde el archivo `functions.php` de un tema, pero sabemos que ese método tiene inconvenientes, así que ahora aprenderemos a crear nuestros propios plugins.

2 Requisitos

2.1 Consideraciones previas

Nombre del plugin

Cuando creamos un plugin debemos elegir un nombre que explique su función pero que no tenga ya otro plugin de los disponibles en el repositorio oficial. De lo contrario WordPress pensará que estamos usando ese otro plugin (y puede intentar actualizarlo). Una solución puede usar incluir un nombre o "alias" de desarrollador que diferencie nuestros plugins del resto.

Prefijar el código

Igual que sucede con el nombre del plugin, es conveniente distinguir nuestras variables y funciones del resto de las que pueda tener una instalación de WordPress. La mejor manera de conseguirlo añadir un prefijo a nuestras funciones a todo nuestro código, desde los archivos a las funciones y variables para evitar posibles colisiones con otros que coincidan en el nombre.

Por ejemplo, el plugin Contact Form 7 antepone la cadena 'wpcf7_' en sus clases y funciones.

Estructura clara

Aunque un plugin puede contar de un único archivo, lo normal es que el plugin se estructure en una carpeta que contenga a su vez otras con los diferentes archivos (.php, .css, .js, imágenes...).

Licencia

Wordpress se distribuye bajo una licencia GNU GPL¹ y todo plugin desarrollado para él debe tener esta licencia u otra que sea compatible con ella. Aunque no es obligatorio, si es conveniente incluir en nuestro plugin un archivo llamado license.php con toda la información del tipo de licencia.

¹ GNU General Public License. <http://www.gnu.org/licenses/gpl.html>

2.2 La cabecera del plugin

El archivo principal de cualquier plugin debe tener un aspecto similar a este:

Ejemplo de cabecera de un plugin

```
<?php
/*
Plugin Name: TelWP Example Plugin
Plugin URI: https://telwp.com/example-plugin
Description: Course plugin example
Version:      20161020
Author:       CursoWP
Author URI:   https://telwp.com
License:      GPL2
License URI:  https://www.gnu.org/licenses/gpl-2.0.html
Text Domain: telwp
Domain Path: /languages
*/
```

Veamos la información contenida en este bloque de comentario²:

- Nombre del plugin. Es el único de los datos que es obligatorio, aunque es conveniente incluirlos todos para aportar la máxima información posible sobre el plugin a los usuarios.
- URI con más información sobre el plugin
- Breve descripción del plugin
- Versión del plugin³
- Nombre del desarrollador o empresa
- Página del desarrollador o empresa
- Licencia y URI con sus términos
- Nombre y ruta del dominio. Se usará para traducir las cadenas de texto de nuestro plugin a otros idiomas.

² Se trata de un DocBlock de PHPDoc.

<https://en.wikipedia.org/wiki/PHPDoc#DocBlock>

PHPDoc es una forma de añadir documentación al código mediante bloques de comentario. Estos pueden ser usados por herramientas como phpDocumentator (<https://phpdoc.org/>) para generar un archivo con la documentación fuera del propio código,

³ https://es.wikipedia.org/wiki/Versi%C3%B3n_de_software

3 Creando un shortcode

Vimos anteriormente como crear tipos de post, taxonomías y campos personalizados. Podremos usar plugins que añadan esa funcionalidad. En este ejemplo, veremos un plugin que genera un shortcode⁴. Un shortcode es una etiqueta especial que los autores pueden usar en el contenido de las entradas/páginas y que será sustituida por el contenido que determinemos.

El autor solo necesita conocer el nombre del shortcode y escribirlo entre corchetes ([]). El texto se guardará en la base de datos de la misma forma en la que lo escribió, pero al rescatar la información pero al mostrar esa página al usuario en la parte pública, ese pequeño código se sustituirá por la cadena que devuelva la función que hayamos asociado⁵. Esa cadena puede incluir código HTML, por lo que los autores podrán añadir cualquier tipo de contenido sin necesidad de saber siquiera HTML.

La función que usemos en el shortcode recibirá tres argumentos. El primero, un array con los posibles atributos que se pasen en la llamada:

⁴ https://codex.wordpress.org/Shortcode_API
<https://developer.wordpress.org/plugins/shortcodes/>

⁵ Es importante devolver la cadena y no imprimirla directamente con una sentencia echo, ya que se mostraría antes que el contenido en lugar de integrada en él

[nombre_shortcode clave1="valor" clave2="otro" ...]

El segundo argumento será el contenido encerrado, en caso de que se use un shortcode de entrada y otro de salida (como si fuera una etiqueta HTML):

[nombre_shortcode]El contenido[/nombre_shortcode]

El tercer argumento será el propio nombre del shortcode, que en nuestro ejemplo no nos hará falta.

```
function telwp_shortcode( $atts, $content=null )
{
    $url = get_permalink( $post->ID );
    $title = get_the_title( $post->ID );
    $tweet = '<a class="twitter-share-button"
href="http://twitter.com/home/?status=Recomendado:
\'\' . $title . \'\' \' . $url . \'\'>
Compartir en Twitter</a>';
    return $tweet;
```

```
}
add_shortcode( 'tweet_this', 'telwp_shortcode' );
```

Como vemos usamos la función `add_shortcode`⁶ para indicar el nombre del shortcode (es decir, la etiqueta que usarán los autores para incluirlo) y la función que se ejecutará.

En esa función hacemos uso de la variable global `$post` para acceder al título y la url del post en el que nos encontremos, y creamos una variable llamada `$tweet`, que será la que devolvamos. Esta variable es una cadena en la que hemos construido un enlace que permite escribir un tweet⁷.

Es un ejemplo sencillo. Si prevemos que vamos a tener títulos largos quizá queramos calcular los caracteres disponibles después del escribir la URL⁸ y el texto de introducción (en el ejemplo hemos puesto “Recomendado: ” antes del título). Si queremos permitir una mayor personalización del mensaje por parte del autor podemos usar las variables `$atts` y `$content`.

Para profundizar en el desarrollo de todo tipo de plugins, uno de los mejores recursos es el Manual de Plugins⁹ oficial.

Otro buen recurso para el aprendizaje es el propio código que controla los shortcodes presentes en el núcleo de WordPress o en otros plugins.

Shortcodes de Wordpress¹⁰:

- `[audio]`¹¹
- `[caption]`¹²
- `[embed]`¹³
- `[gallery]`¹⁴
- `[video]`¹⁵
- `[playlist]`¹⁶

6 https://developer.wordpress.org/reference/functions/add_shortcode/

7 <https://dev.twitter.com/web/tweet-button>

Para simplificar nuestro caso no hemos añadido código Javascript por lo que el enlace no tomará el aspecto del botón de twitter y el enlace se abrirá a pantalla completa y en la misma pestaña.

8 Twitter usa su propio acortador de urls por lo que cualquier enlace, independientemente de su longitud de caracteres, ocupará 23.
<https://support.twitter.com/articles/344685>

9 <https://developer.wordpress.org/plugins/>

10 Puedes consultar la sección “Source File” en la referencia de cada una de sus funciones para ver en que archivo están definidas sus clases

11 https://codex.wordpress.org/Audio_Shortcode

12 https://codex.wordpress.org/Caption_Shortcode

13 https://codex.wordpress.org/Embed_Shortcode

14 https://codex.wordpress.org/Gallery_Shortcode

15 https://codex.wordpress.org/Video_Shortcode

16 https://codex.wordpress.org/Playlist_Shortcode

4 Creando un widget

Vamos a crear otro plugin. En esta ocasión generará un nuevo widget que podremos añadir a nuestras *siderbars*.

WordPress cuenta con una clase para manejar widgets llamada `WP_Widget` y ubicada en `wp-includes/class-wp-widget.php`. Para crear nuestro propio widget crearemos una clase que extienda de ella. Bastará con señalar los parámetros básicos de nuestro widget y pasárselos al constructor de la clase padre.

`wp-content/plugins/cwp-widget/cwp-widget.php`

```
<?php
/*
Plugin Name: TelWP Widget Example Plugin
Version: 0.1
*/

class telwp_widget extends WP_Widget
{
    public function __construct()
```

```
{
    $widget_details = array(
        'classname' => 'telwp_widget',
        'description' => 'Solo un widget'
    );

    parent::__construct( 'telwp_widget', 'Mi
propio widget', $widget_details );
}
}
```

Tendremos que registrar nuestro widget después de que se registren los widgets propios del núcleo. Para ello usaremos la función `register_widget`¹⁷ asociada al *action hook* `widgets_init`¹⁸.

```
add_action( 'widgets_init', function(){
    register_widget( 'telwp_widget' );
}
```

¹⁷ https://developer.wordpress.org/reference/functions/register_widget/

¹⁸ https://developer.wordpress.org/reference/hooks/widgets_init/


```
});
```

Si te resulta mas claro, puedes escribirlo de esta otra forma:

```
add_action( 'widgets_init', 'telwp_widget_init' );

function telwp_widget_init() {
    register_widget( 'TelWP_Widget' );
}
```

Con el widget registrado ya lo tendremos disponible en nuestra sección de widgets¹⁹. Sin embargo, nuestro plugin no mostrará nada en la parte pública salvo un aviso:

function WP_Widget::widget() must be over-ridden in a sub-class.

Sobreescribamos método widget() que se encargue de mostrar algo.

```
class TelWP_Widget extends WP_Widget
{
    public function __construct(){ ... }
    public function widget( $args, $instance ) {
        echo $args['before_widget'];
    }
}
```

¹⁹ Recuerda activar el plugin

```
        echo "¡Este es mi widget!";
        echo $args['after_widget'];
    }
}
```

Los dos parámetros que recibe nuestro método widget están debidamente documentados en el método de la clase padre WP_Widget: \$args es un array que contiene argumentos del área de widgets²⁰ e \$instance es otro array que contiene las propiedades del propio widget. Veamos como podemos usar estas últimas.

La clase WP_Widget cuenta con otros dos métodos que tendremos que sobreescribir para dotar de utilidad a nuestro widget. form() se encargará de mostrar el formulario para configurar el widget, mientras que update() se encargará de actualizar la instancia del widget. Modificaremos también el método widget() para que muestre las propiedades configuradas.

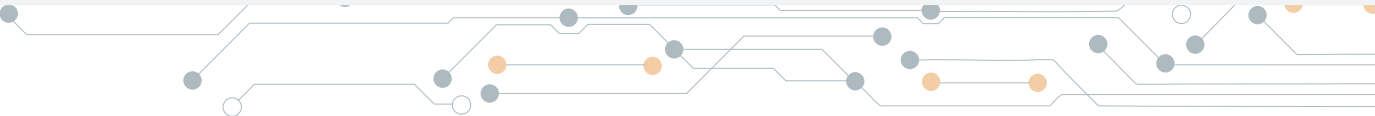
En este caso vamos a tomar el ejemplo directamente de la documentación API de Widgets de WordPress²¹. Este ejemplo muestra como añadir un título al widget. Lo implementaremos y después le añadiremos más funcionalidad.

²⁰ Argumentos como 'before_title', 'after_title', 'before_widget', o 'after_widget' se usan habitualmente en el tema (que es quien define las áreas de widgets) para crear separadores entre los elementos.

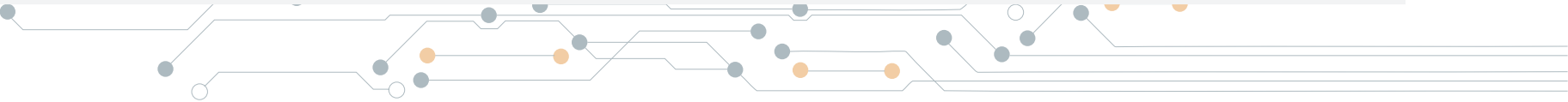
²¹ https://codex.wordpress.org/Widgets_API#Example
Hemos realizado algunas simplificaciones y comentarios extra.

```
class TelWP_Widget extends WP_Widget
{
    public function __construct(){ ... }

    /**
     * Front-end display of widget.
     *
     * @see WP_Widget::widget()
     *
     * @param array $args      Widget arguments.
     * @param array $instance  Saved values from database.
     */
    public function widget( $args, $instance ) {
        echo $args['before_widget'];
        if ( ! $instance['title'] ) { // si existe la propiedad título...
            echo $args['before_title'] . $instance['title'] . $args['after_title']; // la muestra
        }
        echo 'Hello, World!'; // y muestra una cadena (que podría ser cualquier contenido)
        echo $args['after_widget'];
    }
}
```

A decorative graphic at the bottom of the page, resembling a circuit board or a network diagram. It features several interconnected nodes (circles) and lines, with some nodes highlighted in orange and others in grey. The lines are thin and grey, creating a complex web-like structure.

```
/**
 * Back-end widget form.
 *
 * @see WP_Widget::form()
 *
 * @param array $instance Previously saved values from database.
 */
public function form( $instance ) {
    // Muestra el título o "New title" si no hay. esc_html "escapa" las posibles etiquetas HTML
    $title = ! empty( $instance['title'] ) ? $instance['title'] : esc_html( 'New title' );
    ?>
    <p>
    // el campo for del label debe coincidir con el id del input
    <label for="<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>">Título:</label>
    <input class="widefat" id="<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>"
name="<?php echo esc_attr( $this->get_field_name( 'title' ) ); ?>" type="text" value="<?php echo esc_attr(
$title ); ?>" // variables: el id y nombre autogenerados para el campo input y el título asignado antes
    </p>
    <?php
}
```



```
/**
 * Sanitize widget form values as they are saved.
 *
 * @see WP_Widget::update()
 *
 * @param array $new_instance Values just sent to be saved.
 * @param array $old_instance Previously saved values from database.
 *
 * @return array Updated safe values to be saved.
 */
public function update( $new_instance, $old_instance ) {
    $instance = array();
    $instance['title'] = ( ! empty( $new_instance['title'] ) ) ? strip_tags( $new_instance['title']
) : ''; // strip_tags: función de PHP para eliminar etiquetas HTML y PHP que pueda tener la cadena.

    return $instance;
}

} // cierre de class TelWP_Widget
```

Si acudimos ahora a la sección de widgets y añadimos nuestra creación, podremos definir un título. `form()` se encarga de pintarlo y `update()` de procesarlo.

Y al consultar el widget desde la parte pública veremos el resultado del método `widget()` que muestra el título y un saludo. Piensa que ese saludo podía ser sustituido por cualquier cosa. De hecho es lo que vamos a hacer ahora. Asegúrate de comprender el código realizado hasta ahora antes de continuar.

Uno de los widgets por defecto de WordPress permite mostrar una lista con los títulos de últimos posts. Vamos a crear un widget muy parecido pero que nos permita seleccionar los últimos post pero de una categoría determinada que podremos seleccionar entre las que tengamos definidas en nuestra web.

Para hacer esto, usaremos una función llamada `wp_dropdown_categories`²² que como su nombre indica nos crea un *dropdown* (etiqueta "select" de HTML) con todas las categorías. Esta función acepta un array con el que podremos definir muchos parámetros del select que generará como el valor por defecto, o los atributos 'id' y 'name'.

Y para rescatar de la base de datos los posts específicos de la categoría seleccionada desde el administrador, haremos uso de una clase que ya conocemos: `WP_Query`. Con ella obtendremos, filtrando los posts de nuestro interés, un array que almacenaremos

en una variable²³ y recorreremos para llenar nuestra lista. En caso de que el array esté vacío evitaremos mostrar siquiera el título del widget.

A continuación puedes consultar el código modificado para los tres métodos: `widget()`, `form()` y `update()`. No hemos modificado el constructor, pero sería recomendable hacerlo para cambiar la descripción del widget por una que refleje su nuevo comportamiento²⁴.

Si comparas el resultado que se obtiene con nuestro código con el widget de WordPress "Entradas recientes" (*Recent Posts*) verás que nosotros no hemos dado al administrador la opción de mostrar las fechas de las entradas junto los títulos. Puedes probar a añadir esta y otras funcionalidades para hacer el plugin más flexible²⁵. Ten en cuenta que el mismo plugin, con diferente configuración, puede ser usado más de una vez en las diferentes áreas de widgets.

²² https://developer.wordpress.org/reference/functions/wp_dropdown_categories/

²³ A la que no llamaremos `$wp_query` para evitar conflictos con la consulta principal de la página en la que se muestre el widget.

²⁴ Del mismo modo debería cambiarse el propio nombre del widget y su clase por uno más representativo.

²⁵ Puedes tomar como base el código de otros plugins, incluidos los propios de WordPress ubicados en `wp-includes/widgets`.

```

public function widget( $args, $instance ) {
    $r = new WP_Query( apply_filters( 'widget_posts_args', array(
        'cat'                => $instance['cat'],
        'posts_per_page'     => $instance['qty'],
        'post_status'        => 'publish'
    ) ) );
    if ( $r->have_posts() ) :
        echo $args['before_widget'];
        if ( $instance['title'] ) {
            echo $args['before_title'] . $instance['title'] . $args['after_title'];
        } ?>
        <ul>
        <?php while ( $r->have_posts() ) : $r->the_post(); ?>
            <li><a href="<?php the_permalink(); ?>"><?php get_the_title() ? the_title(); ?></a></li>
        <?php endwhile; ?>
        </ul>
        <?php echo $args['after_widget'];
        wp_reset_postdata(); // Reseteamos la variable global $the_post al acabar
    endif;
}

```

```

public function form( $instance ) {
    $title = ! empty( $instance['title'] ) ? $instance['title'] : 'Últimamente';
    $select = ! empty( $instance['cat'] ) ? $instance['cat'] : 0;
    $qty = ! empty( $instance['qty'] ) ? $instance['qty'] : 5;
    ?>
    ... // campo de título sin modificar
    <p>
    <label for="<?php echo $this->get_field_id( 'cat' ); ?>">Categoría</label>
    <?php wp_dropdown_categories( [ 'selected' => $select,
                                'id' => $this->get_field_id( 'cat' ),
                                'name' => $this->get_field_name( 'cat' ) ] ); ?>

    </p>
    <p>
    <label for="<?php echo $this->get_field_id( 'qty' ); ?>">Cantidad</label>
    <input class="tiny-text" id="<?php echo $this->get_field_id( 'qty' ); ?>" name="<?php echo $this->get_field_name( 'qty' ); ?>" type="number" step="1" min="1" value="<?php echo $qty; ?>" size="2">
    </p>
    <?php
}

```

```
public function update( $new_instance, $old_instance ) {  
    $instance = array();  
    $instance['title'] = ( ! empty( $new_instance['title'] ) ) ? strip_tags( $new_instance['title'] )  
: '';  
    $instance['cat'] = intval($new_instance['cat']);  
    $instance['qty'] = intval($new_instance['qty']);  
  
    return $instance;  
}
```

La complejidad del código dependerá de las tareas que queramos realizar en él²⁶, pero para crear un widget basta con extender una clase y sobrescribir 4 métodos (incluyendo el constructor).

Recuerda que puedes consultar más información y ejemplos sobre la API para manejar widgets está disponible en su sección específica de la documentación oficial²⁷

²⁶ Y de que hagamos un código claro y ordenado. Generalmente se acusa a WordPress de no ser un buen ejemplo en ese sentido, pero nada nos impide intentar mejorarlo. Por ejemplo, creando métodos privados a los que extraer parte de la lógica, facilitando la reutilización y la lectura del código.

²⁷ https://codex.wordpress.org/Widgets_API

Telefonica

EDUCACIÓN DIGITAL