WordPress Plugins **Feb 12, 2025** **Deyimar A.** **14min de lectura**

Cómo crear un plugin de WordPress

WordPress dispone de una enorme colección de plugins que introducen funcionalidades personalizadas. Es más, los plugins pueden añadir nuevas funciones a tu sitio web de WordPress sin tener que cambiar el código principal.

Aunque hay toneladas de plugins gratuitos y premium entre los que elegir, puede haber ocasiones en las que necesites funciones específicas de WordPress que no estén disponibles. Para ello, es posible que tengas que crear tu propio plugin de WordPress.

En este tutorial aprenderás los pasos para crear un plugin de WordPress. También repasaremos las mejores prácticas y normas de programación y creación de plugins.

Además, este artículo cubrirá las diferencias entre un plugin y un tema y cómo funcionan en la plataforma WordPress.

Tabla de Contenidos

¿Qué necesitas para hacer un plugin de WordPress?

- Editor de texto
- **Acceso FTP** a tu cuenta de hosting
- Una **instalación de WordPress** que funcione

Necesitarás un editor de texto para escribir el código del plugin. Algunos de los **editores de HTML** más populares son **Notepad++** y **Atom.**

Después de instalar el editor de texto, conéctalo a tu servidor FTP para modificar el código. Puedes aprender a conectar Notepad++ a tu servidor FTP leyendo nuestro tutorial en inglés sobre **cómo conectarte a FTP con Notepad++**.

Luego, configura un cliente FTP para subir el archivo del plugin a tu sitio web. Recomendamos utilizar la aplicación **FTP FileZilla**, ya que es fácil de configurar.

Por último, asegúrate de que tienes una instalación de WordPress que está actualizada y funciona. Hay varias formas de actualizar los archivos del núcleo de WordPress si has desactivado las actualizaciones automáticas. **Haz una copia de seguridad de tus archivos de WordPress** antes de actualizar el sitio para evitar la pérdida de datos.

Como alternativa, considera la posibilidad de **instalar WordPress localmente**. Este método no requiere un sitio web en vivo con un nombre de dominio y un plan de alojamiento, por lo que puedes probar tu plugin en WordPress sin que los visitantes lo vean de inmediato.

Tener un conocimiento básico de PHP beneficiará tu proceso de desarrollo de plugins. Tendrás que escribir una función personalizada y llamar a funciones existentes del núcleo de WordPress. Como mínimo, deberías estar familiarizado con las convenciones de nomenclatura de PHP y la estructuración de archivos.

Plugins de WordPress vs Temas de WordPress

La funcionalidad del sitio de WordPress puede modificarse mediante plugins y temas.

Los temas de WordPress tienen un archivo **functions.php** almacenado en la carpeta **/wp-includes/**, que te permite añadir código personalizado para nuevas funciones.

Aunque este método funciona para pequeñas alteraciones, es poco práctico para implementar cambios importantes que afecten a todo el sitio web.

Esto se debe a que la funcionalidad almacenada en el archivo **functions.php** depende de si el tema está activo o no. Desactivar el tema de WordPress revertirá los cambios realizados en dicho archivo y provocará un error cuando el sitio llame a las funciones que faltan.

A menos que utilices un tema hijo, la actualización del tema también sobrescribirá el archivo **functions.php**, obligándote a restaurar manualmente el código personalizado de WordPress.

Por eso es útil crear un plugin personalizado. Hacerlo facilita la modificación del comportamiento predeterminado de WordPress para adaptarlo a tus necesidades.

Puedes añadir plugins de WordPress a cualquier instalación de WordPress. Las funciones introducidas por el plugin seguirán funcionando aunque cambies de tema. Además, las actualizaciones no sobrescribirán las funciones existentes, lo que te ahorrará tiempo y esfuerzo.

¿Qué son los hooks de WordPress?

Los Plugins de WordPress interactúan con código de núcleo utilizando **hooks**, que en español se traducirían como ganchos. Hay dos tipos diferentes de hooks.

1. **Hooks de acción**: sirven para agregar o quitar funciones.
2. **Hooks de filtro**: sirven para modificar datos arrojados por funciones.

script de hook de acción **wp_head()** antes de la etiqueta de cierre **(</head>)** de cualquier página.

Los ganchos de acción son contextuales: algunos se llaman en cada página de tu sitio web, otros sólo se llaman cuando se visualiza el Panel de control del administrador, etc. Se puede encontrar una lista completa de los hooks de acciones y el contexto en el que se llaman en la página **WordPress Plugin API/Action Reference**.

Añadiendo funciones a un hook de acción usando add_action()

Para agregar funciones a un hook de acción en un archivo de plugin se debe llamar a la función **add_action()** con al menos dos parámetros.

```
// Hook to the 'init' action, which is called after WordPress is finished loading the core code
add_action( 'init', 'add_Cookie' );

// Set a cookie with the current time of day
function add_Cookie() {
  setcookie("last_visit_time", date("r"), time()+60*60*24*30, "/");
}
```

El primer parámetro es el nombre del **hook de acción** que quieres adjuntar la llamada de retorno, mientras que el segundo parámetro contiene el nombre de la **función** que quieres ejecutar.

El tercer parámetro (opcional) es la **prioridad** de la función que deseas ejecutar. La prioridad predeterminada es 10, poniendo tu función personalizada después de cualquiera de las funciones integradas de WordPress.

El cuarto parámetro (opcional) es el **número de argumentos**, lo que significa cuántos parámetros tu función personalizada puede tomar. El valor predeterminado es 1.

Ejemplo de código de plugin para mostrar texto después del pie de cada página

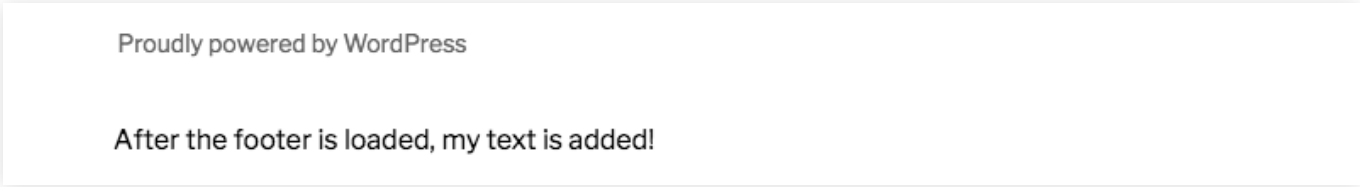
Este complemento asocia el hook de acción **wp_footer()**, que se llama justo antes de la etiqueta **</body>** de cierre de cada página, y añade una nueva función llamada **mfp_Add_Text()**. Puesto que esto es parte de un plugin y no del tema, continuará funcionando así actives un tema totalmente diferente.

Puedes guardar este ejemplo como un archivo PHP, cargarlo en la carpeta de **plugins**.

```
<?php
/*
Nombre del Plugin: Añadir Texto Al Pie de la Página
*/
```

```
function mip_add_text()
{
  echo "<p style='color: black;'>After the footer is loaded, my text is added!</p>";
}
```

La siguiente captura de pantalla muestra el plugin en acción después de activarlo a través del panel de administración de WordPress:



> **¡Importante!:** PHP evalúa todo el script antes de ejecutarlo. Escribir las llamadas **add_action()** en la parte superior del archivo en el orden en que se ejecutan y, a continuación, definir tus funciones en el mismo orden, hace que el archivo sea más fácil de leer.

Eliminando funciones de un hook de acción usando remove_action()

Para **eliminar** una acción de un hook de acción, debes escribir una nueva función que llame a **remove_action()** y, a continuación, llama a la función que has escrito utilizando **add_action()**.

La función **remove_action()** debe contener al menos dos parámetros.

```
// Hook the 'init' action, which is called after WordPress is finished loading the core code, add the function 'remove_My_Meta_Tags'
add_action( 'init', 'remove_My_Meta_Tags' );

// Remove the 'add_My_Meta_Tags' function from the wp_head action hook
function remove_My_Meta_Tags()
{
  remove_action( 'wp_head', 'add_My_Meta_Tags');
}
```

El primer parámetro requerido es el nombre del **hook de acción** al que está asociada la función mientras que el segundo parámetro requerido es el nombre de la **función** que deseas eliminar.

Una forma de hacerlo es utilizar la función PHP date() para obtener el día actual, seguida de etiquetas condicionales para comprobar si es lunes. Después de analizar la información, la página ejecutará la función remove_action() en todos los posts publicados los lunes.

En el siguiente ejemplo, evitaremos que el texto extra agregado al pie de página aparezca en las entradas de los lunes.

Una forma de hacerlo es utilizando la función fecha de PHP para obtener el día actual, seguido de una instrucción condicional para probar si el día actual es lunes. Después de analizar la información, la página ejecutará la función **remove_action ()** en todos los posts publicados los lunes.

```
<?php

// Hook the 'wp_footer' action, run the function named 'mfp_Add_Text()'
add_action("wp_footer", "mfp_Add_Text");

// Hook the 'wp_head' action, run the function named 'mfp_Remove_Text()'
add_action("wp_head", "mfp_Remove_Text");

// Define the function named 'mfp_Add_Text('), which just echoes simple
text
function mfp_Add_Text()
{
  echo "<p style='color: #FFF;'>After the footer is loaded, my text is
added!</p>";
}

// Define the function named 'mfp_Remove_Text()' to remove our previous
function from the 'wp_footer' action
function mfp_Remove_Text()
{
  if (date("l") === "Monday") {
    // Target the 'wp_footer' action, remove the 'mfp_Add_Text' function
from it
    remove_action("wp_footer", "mfp_Add_Text");
  }
}
```

Filtros y hooks de filtro

Un filtro es una función PHP llamada por un hook de filtro específico que modifica los datos devueltos por las funciones existentes. Al igual que los hooks de acción, los hooks de filtro también son contextuales.

Preguntarle a Kodee

Para **agregar** una función de filtro a cualquier gancho de filtro, tu complemento debe llamar a la función de WordPress **add_filter()**, con al menos dos parámetros.

```
// Hook the 'the_content' filter hook (content of any post), run the function named 'mfp_Fix_Text_Spacing'
add_filter("the_content", "mfp_Fix_Text_Spacing");

// Automatically correct double spaces from any post
function mfp_Fix_Text_Spacing($the_Post)
{
 $the_New_Post = str_replace("  ", " ", $the_Post);

 return $the_New_Post;
}
```

El primer parámetro requerido es el nombre del **hook del filtro** que quieres asociar, mientras que el segundo parámetro contiene el nombre de la **función** que quieres que se ejecute cuando se aplique el filtro.

El tercer parámetro (opcional) es la **prioridad** de la función que deseas ejecutar. Puedes asociar cualquier cantidad de funciones de filtro diferentes a un mismo hook de filtro. La prioridad predeterminada es 10, poniendo tu función personalizada después de cualquier función incorporada.

El cuarto parámetro (opcional) es el **número de argumentos**, lo que significa cuántos parámetros tu función de filtro personalizado será capaz de tomar. El valor predeterminado es 1.

Ejemplo de plugin para modificar el extracto de una publicación

WordPress tiene una función que arroja el extracto de una entrada, está función se llama **get_the_excerpt()**, que también es un hook de filtro. Esta función realmente lo que hace es mostrar el extracto que **get_the_excerpt()** arroja, es ahí donde se aplica el filtro y el fragmento se altera antes de ser mostrado.

El siguiente ejemplo de plugin define una función de filtro que toma el extracto como su único parámetro de entrada, añade un texto antes de él y devuelve el nuevo valor cada vez que el script llama a la función **get_the_excerpt().**

Como el valor devuelto de la función **get_the_excerpt()** es el texto del extracto real, el plugin introducirá automáticamente el nuevo valor como parámetro de la función **$old_Excerpt** cuando se llame usando **add_filter()**. La función que definas debe devolver el nuevo valor.

```
<?php
/*
Plugin Name: Add Excerpt
```

```
// Take the excerpt, add some text before it, and return the new excerpt
function mfp_Add_Text_To_Excerpt($old_Excerpt)
{
  $new_Excerpt = "<b>Excerpt: </b>" . $old_Excerpt;
  return $new_Excerpt;
}
```

Eliminando filtros mediante remove_filter ()

Eliminar un filtro es mucho más sencillo que eliminar una acción, ya que WordPress te permite llamar a la función **remove_filter()** sin necesidad de definir una nueva función.

En el siguiente ejemplo, eliminaremos el texto adicional del extracto si el día actual es jueves. Utilizaremos la función **remove_filter()** con al menos dos parámetros.

El primero debe contener el hook del filtro al que se adjunta la función. El segundo parámetro debe ser el nombre del filtro que quieres eliminar. Añade un parámetro de prioridad si lo has definido al crear la función.

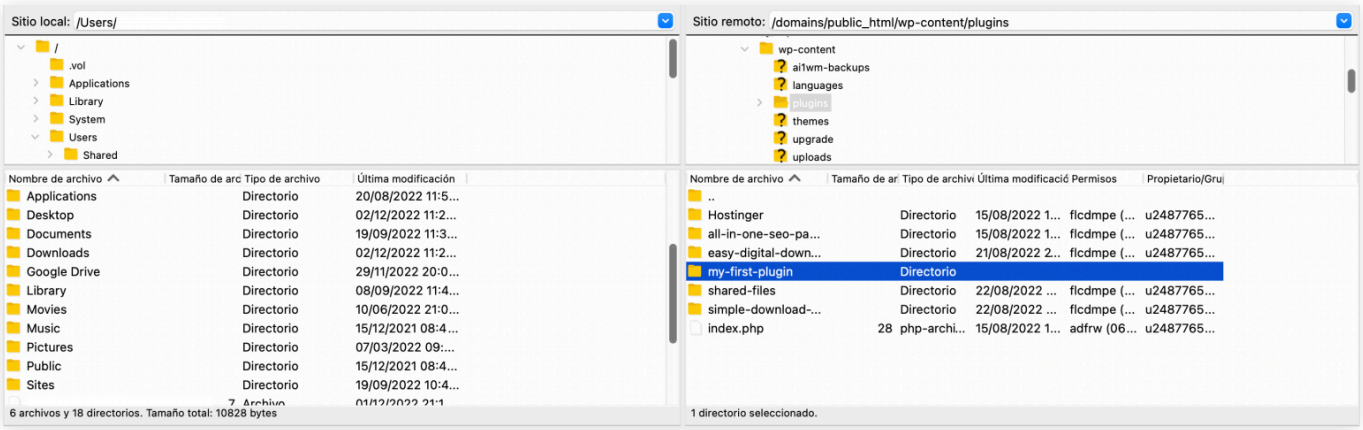
```
// Hook the get_the_excerpt filter hook, run the function named
mfp_Add_Text_To_Excerpt
add_filter("get_the_excerpt", "mfp_Add_Text_To_Excerpt");

// If today is a Thursday, remove the filter from the_excerpt()
if (date("l") === "Thursday") {
  remove_filter("get_the_excerpt", "mfp_Add_Text_To_Excerpt");
}

// Take the excerpt, add some text before it, and return the new excerpt
function mfp_Add_Text_To_Excerpt($old_Excerpt)
{
  $new_Excerpt = "<b>Excerpt: </b>" . $old_Excerpt;
  return $new_Excerpt;
}
```

Ahora que tienes una comprensión básica de los ganchos y los filtros, vamos a crear un plugin simple de WordPress que añade un nuevo enlace y una página al Panel de control del administrador.

¡Importante! Utilizar un sitio de staging de WordPress para probar nuevos plugins te ayudará a evitar errores que puedan causar un tiempo de inactividad. Hay dos maneras de **crear un entorno de staging**: manualmente o utilizando un plugin como WP Staging. También puedes instalar WordPress localmente en tu ordenador.

asegurarte de que el nuevo nombre no está ya en uso.

Utiliza **un cliente FTP** para conectarte a tu cuenta de alojamiento y facilitar el proceso de subida de archivos. Desde el directorio principal de WordPress, navega a **wp-content -> plugins**. Dentro de la **carpeta de plugins**, crea una nueva carpeta llamada **my-first-plugin** (“mi primer plugin” en español):



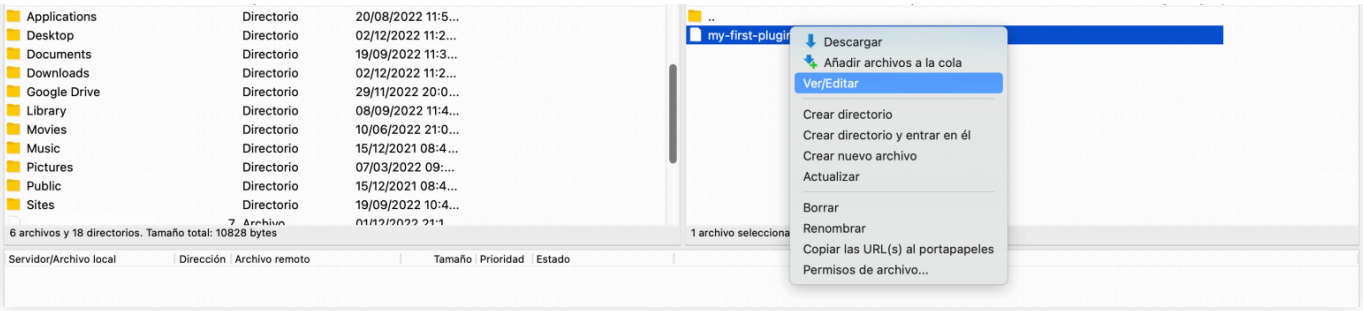
Practicar la gestión de archivos durante el desarrollo de WordPress facilitará mucho el proceso a largo plazo. Por esto, te recomendamos separar los distintos archivos que componen tu plugin de WordPress en subcarpetas según su funcionalidad.

Por ejemplo, guarda los archivos CSS, PHP y JavaScript en carpetas separadas. A medida que desarrolles tu plugin de WordPress personalizado, será más fácil localizar archivos específicos cuando todo tenga un directorio dedicado.

Paso 2 – Creación del primer archivo

El archivo principal del plugin contendrá la información que WordPress necesita para mostrar tu plugin en la lista de plugins, donde podrás activarlo.

Crea un nuevo archivo PHP llamado **my-first-plugin.php** en la carpeta que has creado antes. Este archivo principal del plugin contendrá comentarios de cabecera con información adicional para que WordPress la lea o muestre.

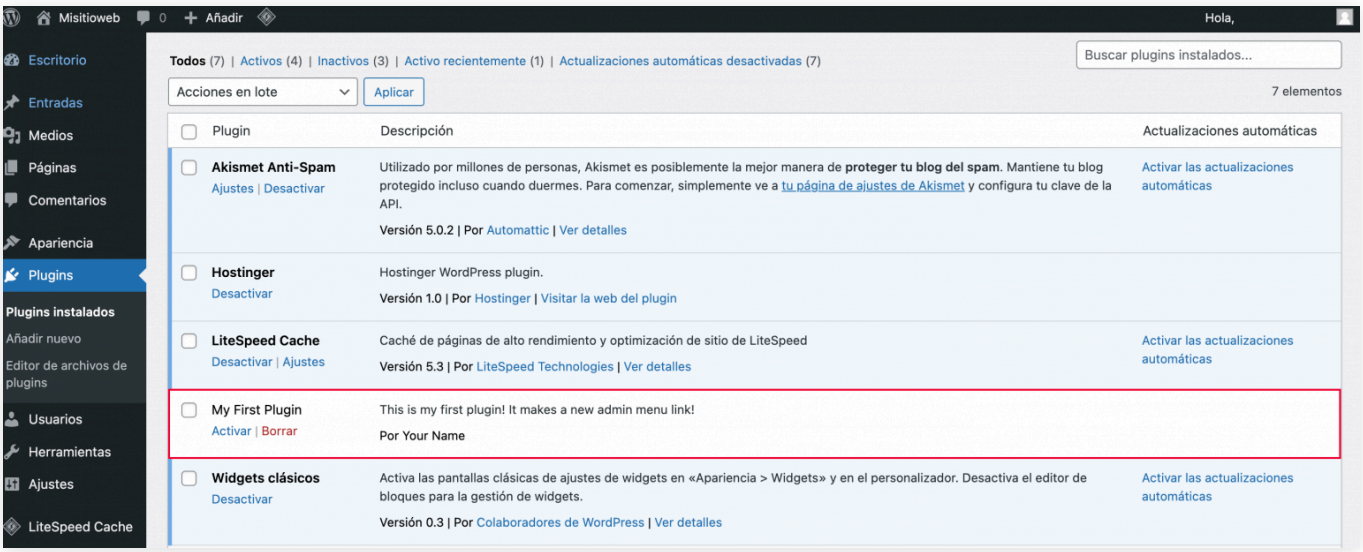


A continuación, haz clic con el botón derecho en el archivo y selecciona **Ver/Editar** para añadir el siguiente código mediante un editor de HTML:

```
<?php
/*
Plugin Name: My First Plugin
Description: This is my first plugin! It makes a new admin menu link!
Author: Your Name
*/
```

Puedes consultar este **manual de PHP** para entender por qué la etiqueta de cierre **?>** no es necesaria aquí.

Guarda el archivo. A continuación, navega a la sección de **plugins** de tu panel de control de WordPress. Si WordPress ha leído correctamente el nuevo archivo, verás **My First Plugin** en la lista:



Paso 3 – Escribiendo las funciones d

En la carpeta principal de tu plugin, crea una nueva carpeta llamada **includes**. Lo utilizaremos para almacenar los archivos de apoyo utilizados por el archivo principal. En esta carpeta, crea un nuevo archivo PHP y guárdalo como **mfp-functions.php**. Debes declarar la apertura con la etiqueta **<?php** en la primera línea.

Este nuevo archivo es donde se almacenarán todas las funciones de tu plugin.

Ahora regresa a **my-first-plugin.php,** en la carpeta principal de tu plugin. Debes incluir el archivo **mfp-functions.php** para que puedas usar las nuevas funciones. Utiliza **require_once** para asegurarte de que el plugin solo funciona si el archivo de funciones está disponible.

Edita **my-first-plugin.php** como se muestra a continuación, luego guárdalo y cárgalo una vez más, sobrescribiendo la versión anterior cuando se te pregunte.

```
<?php
/*
Plugin Name: My First Plugin
Description: This is my first plugin! It makes a new admin menu link!
Author: Your Name
*/

// Include mfp-functions.php, use require_once to stop the script if mfp-functions.php is not found
require_once plugin_dir_path(__FILE__) . 'includes/mfp-functions.php';
```

La función de WordPress **plugin_dir_path(FILE)** te permite incluir archivos de tu carpeta de plugins, dando la ruta completa al directorio que almacena el nuevo plugin.

Ahora, vuelve al archivo **mfp-functions.php** en el directorio **Includes**. Debido a que nuestro plugin añadirá un nuevo enlace de nivel superior al menú de navegación del panel de control del administrador, utilizaremos una función personalizada llamada **mfp_Add_My_Admin_Link()**. Añade el siguiente bloque de código al archivo **mfp-functions.php**:

```
<?php
/*
 * Add my new menu to the Admin Control Panel
 */
// Hook the 'admin_menu' action hook, run the function named 'mfp_Add_My_Admin_Link()'
add_action( 'admin_menu', 'mfp_Add_My_Admin_Link' );
// Add a new top level menu link to the ACP
function mfp_Add_My_Admin_Link()
{
      add_menu_page(
        'My First Page', // Title of the page
```

```
}
```

> **¡Importante!** Agrupa las funciones similares y añade una descripción sobre cada una de ellas mediante un comentario de varias líneas. Esto facilitará las futuras actualizaciones y la depuración del plugin.

mfp_Add_My_Admin_Link() utiliza la función integrada de WordPress **add_menu_page()** con al menos cuatro parámetros en el siguiente orden:

- **Título de la página**: el nombre de la página que se muestra en la pestaña del navegador.
- **Título del menú**: el texto utilizado para el elemento del menú. En nuestro ejemplo, es el nombre del plugin.
- **Capacidad**: requisito de capacidad del usuario para ver el menú del plugin. Aquí, solo los usuarios con la capacidad **manage_options** pueden acceder a la página enlazada.
- **Slug del menú**: el archivo que se utilizará para mostrar la página real. Crearemos el archivo **mfp-first-acp-page.php** vinculado en la carpeta **Includes** en la siguiente sección.
- **Función (opcional)**: la función que produce el contenido de la página.

Adjuntar la función personalizada mediante **add_action()** permite que el plugin llame al hook de acción en determinadas circunstancias. Añadir **admin_menu** como primer parámetro llamará a la función cuando un usuario acceda al menú de administración. Mientras tanto, **mfp_Add_My_Admin_Link** es la función que se ejecutará cuando se especifique como segundo parámetro.

Por último, sube el archivo del plugin **mfp-functions.php** a la carpeta **Includes**.

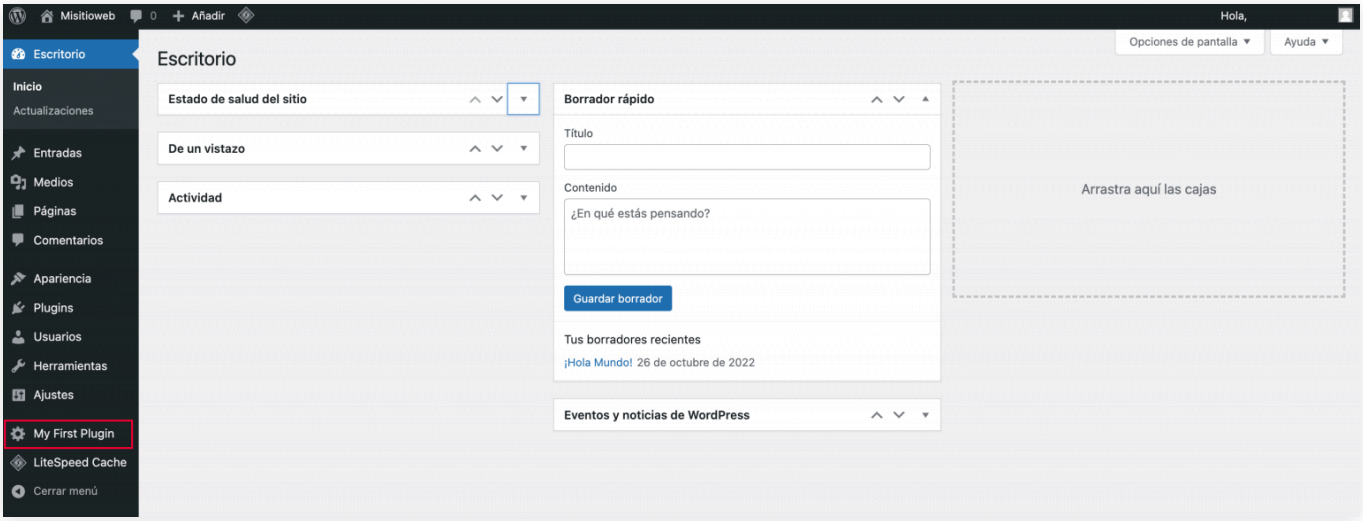
Paso 4 – Creación de la nueva página de administración

Después de definir las funciones del plugin, es hora de construir la página a la que nos llevará el botón del menú. Crea un nuevo archivo PHP llamado **mfp-first-acp-page.php** en la subcarpeta **Includes** y añade el siguiente código:

```
<div class="wrap">
  <h1>Hello!</h1>
  <p>This is my plugin's first page</p>
</div>
```

Al crear las páginas de administración, WordPress recomienda adjuntar tu propio HTML con una etiqueta ** ** y darle la clase **"wrap"**, como se muestr

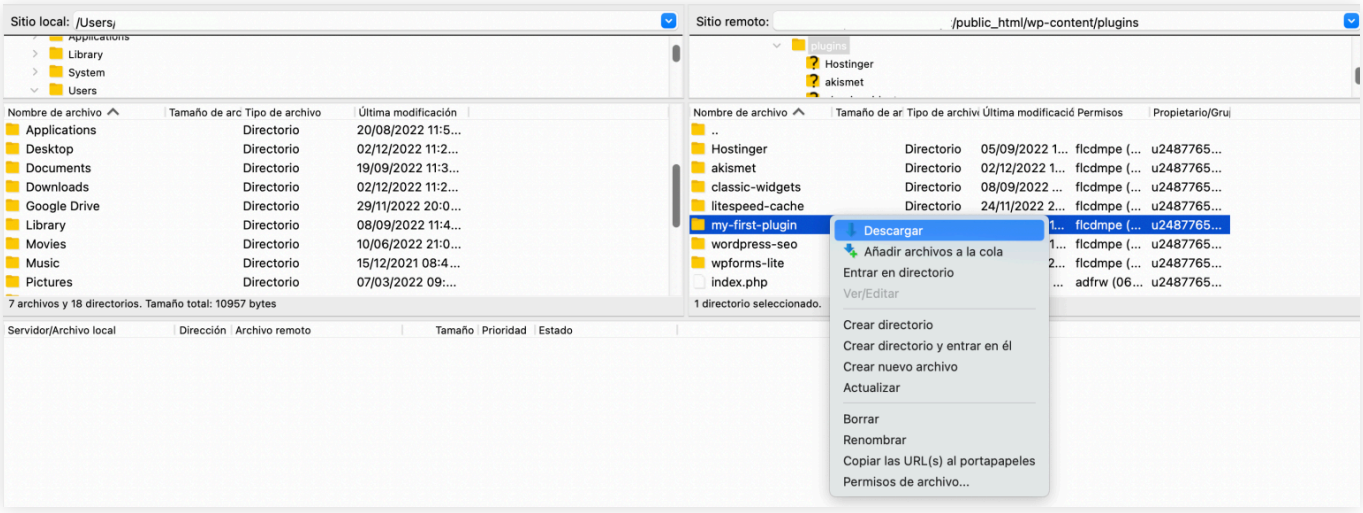
plugin aparecerá en la parte inferior del menú de navegación.



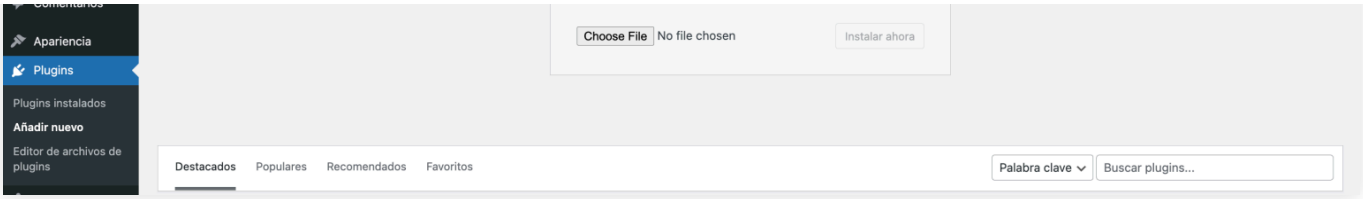
¡Enhorabuena! Has creado con éxito tu primer plugin de WordPress.

Si has creado el plugin en un sitio de staging, tendrás que **instalar el plugin** en el sitio real. Los siguientes pasos te mostrarán cómo hacerlo:

1. En FileZilla, haz clic con el botón derecho en la carpeta de **my-new-plugin** y selecciona **Descargar**. A continuación, comprime los archivos en un archivo ZIP.



2. Ve al menú de **plugins** desde tu panel de control de WordPress. A continuación, haz clic en **Añadir nuevo**.
3. Haz clic en **Subir Plugin** y selecciona el archivo ZIP de tu plugin.



4. Selecciona **Instalar ahora** para iniciar el proceso de instalación.

Lecturas recomendadas

¿Te preguntas en qué idiomas están escritos los plugins de WordPress? Aprende más en las siguientes guías:
¿Qué es HTML?
¿Qué es CSS?
¿Qué es JavaScript?

Grandes normas y prácticas al crear plugins personalizados

A medida que las necesidades de tu sitio evolucionen continuamente, tendrás que revisar el código del plugin para implementar actualizaciones y parches de seguridad.

Teniendo esto en cuenta, sigue las mejores prácticas para el desarrollo de plugins desde el principio. Si lo haces, todo el proceso será más fácil para ti y para los desarrolladores web con los que puedas trabajar en el futuro.

Además, consulta los **mejores ejemplos de plugins de WordPress** para inspirarte. Fíjate en su código fuente, en cómo organizan sus carpetas y en otras prácticas que debes aplicar al crear plugins de WordPress.

Aquí tienes algunas de las mejores prácticas de codificación y desarrollo de plugins para ayudarte en la creación de tu primer plugin de WordPress:

- **Desarrolla y prueba los plugins de WP en un entorno de staging.** De este modo, no habrá riesgo de romper el sitio si un plugin tiene un código defectuoso.
- **Construye una estructura de carpetas lógica.** Crea sub carpeta
  divide el código en archivos separados según su finalidad o tipo d

- **Crea documentación.** Esta práctica es especialmente beneficiosa si creas plugins con funcionalidades complejas para un gran número de usuarios.
- **Utiliza un software de control de versiones para hacer un seguimiento de los cambios realizados en tu código.** Saber quién ha añadido algo al código, ayudará a evitar choques entre actualizaciones y a reducir el número de errores.
- **Consulta el Codex de WordPress para conocer las normas de codificación específicas del idioma.** Asegúrate de cumplirlas cuando colabores en un proyecto.
- **Activa WP_DEBUG o utiliza una herramienta de depuración cuando desarrolles plugins.** Hacerlo facilitará la localización de errores, acelerando el proceso general de construcción de plugins.

Conclusión

Desarrollar un plugin personalizado es una forma de añadir funcionalidad a un sitio de WordPress que los plugins disponibles actualmente no ofrecen. Puede ser un plugin sencillo que implemente pequeñas alteraciones o uno complejo que modifique todo el sitio.

Para recapitular, estos son los pasos para crear un plugin de WordPress desde cero:

1. Crear una carpeta para almacenar los archivos del plugin.
2. Crear el archivo principal de tu plugin.
3. Añadir código a varios archivos para las funciones del plugin.
4. Construir la página de administración del plugin.

Como con cualquier otra habilidad, se necesita tiempo para ser bueno en la creación de plugins de WordPress. Con suficiente práctica, podrás crear plugins y ponerlos a disposición para su descarga en el directorio de plugins de WordPress o incluso venderlos en alguno de los marketplaces.

Esperamos que este artículo te haya enseñado a crear un plugin para WordPress. Si tienes alguna pregunta u observación, no dudes en dejar un comentario.

Cómo crear un plugin para WordPress – FAQ

¿Puedes ganar dinero haciendo plugins para WordPress?

Sí, puedes vender plugins a través de tu propio sitio web o en u

¿Los plugins de WordPress están escritos en PHP?

Los plugins de WordPress suelen estar escritos en PHP, pero también necesitarás saber algo de HTML y CSS básicos para gestionar correctamente la salida del plugin.

Todo el contenido de los tutoriales en este sitio web está sujeto a los rigurosos estándares y valores editoriales de Hostinger.

EL AUTOR

Deyimar Albornoz

Deyi es una entusiasta del marketing digital, con experiencia en diseño de

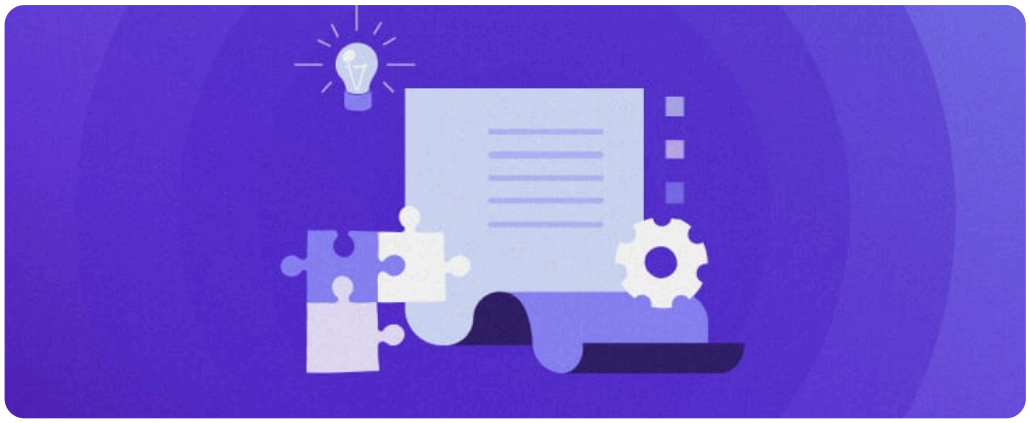
Tutoriales relacionados

28 Nov • WORDPRESS

Cómo despublicar un sitio, páginas o entradas de WordPress

Como usuario de WordPress, puede que a veces quieras eliminar tu sitio web de WordPress, páginas o entradas específicas de forma permanente. Otras...

Por Federico Foscarini

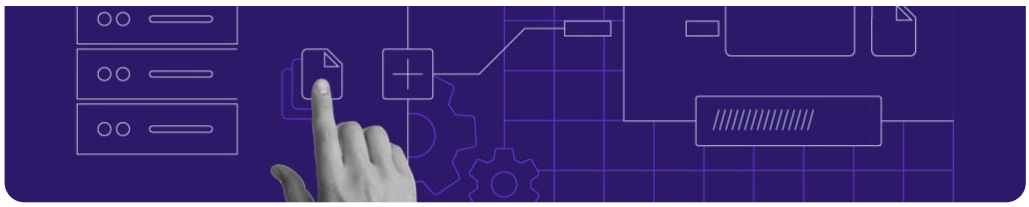


20 Nov • WORDPRESS

“Sorry you are not allowed to access this page”: 10 formas de solucionar el error en WordPress

Mientras trabajas en tu sitio web WordPress, hay pocas cosas más frustrantes que encontrarse con el error “sorry you are not allowed to access this...

Por Carlos Mora



14 Nov • WORDPRESS

Cómo exportar entradas de WordPress con imágenes

Puedes exportar entradas de WordPress, incluyendo sus imágenes, usando la herramienta nativa de Exportar (aunque requiere un paso manual para las...

Por Diego Boada

Lo que dicen nuestros clientes