Según la definición de WordPress.com, un código corto o shortcode es un código específico de WordPress que le permite hacer cosas ingeniosas con muy poco esfuerzo. Los WordPress shortcodes pueden embeber archivos o crear objetos que normalmente requieren mucho código complicado y feo en una sola línea. En otras palabras, gracias a los WordPress shortcodes no hay necesidad de que los usuarios añadan manualmente código HTML al contenido de la publicación y además, es posible cambiar dinámicamente el output bajo condiciones específicas (es decir, el usuario conectado, la fecha actual, ubicación del usuario, etc.).

Si usted alguna vez ha usado la [galería] de Shortcodes, entonces ya tiene una idea de cómo los WordPress Shortcodes funcionan:

```
[gallery ids="129,130,131,132" order="DESC" orderby="title" columns="4"]
```

En este ejemplo, galería es la etiqueta de shortcode, mientras que ids, order, orderby y colums son los atributos de shortcode. Estos atributos determinan el código HTML devuelto.

Tipologías y Estructura de los Shortcodes

WordPress proporciona dos tipologías de shortcodes:

Los self-closing shortcodes se ven como la [galería] de shortcodes, y no requieren una etiqueta para cerrar.
Los enclosing shortcodes requieren una etiqueta de cierre, y permiten la manipulación del contenido cerrado.

He aquí un ejemplo de inclusión de Shortcode:

```
[gist]2314628[/gist]
```

Este shortcode es proporcionado por el plugin de Jetpack para embeber en publicaciones el código de un Gist público especificado por su ID. La cadena de texto entre la etiqueta de apertura y cierre es el contenido de shortcode.

Lo que Aprenderá en Esta Publicación

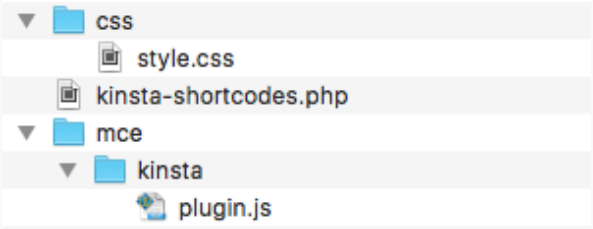
De inmediato WordPress proporciona seis shortcodes, y plugins que puede agregar mucho más gracias a la API de Shortcode .

La Api de Shortcode es un conjunto simple de funciones para crear WordPress shortcodes para su uso en publicaciones y páginas.

En esta publicación les presentaré esta función útil de WordPress desde la perspectiva de un desarrollador. A partir de ahora, nos adentraremos en la API de Shortcode siguiendo estos pasos:

- Una introducción a la construcción de un shortcode
- Un self-closing shortcode
- Un enclosing shortcode
- Cómo incluir shortcodes sacando provecho de TinyMCE UI
- Usos no convencionales de shortcodes

El código mostrado en los siguientes ejemplos se ha agrupado en un plugin disponible para descargar en Github.



La imagen muestra la estructura del plugin de Shortcodes de Kinsta

WordPress Shortcodes para Desarrolladores

Cuando WordPress procesa el contenido de la publicación, busca shortcodes registrados. Una vez que se encuentre un shortcode, el Shortcode API analiza la etiqueta, los atributos y el contenido cerrado, si está disponible, y los pasa a una función de manipulador correspondiente. Esta función realiza una tarea y devuelve una cadena para incluirla en el contenido de la publicación. Por lo tanto, nuestro primer paso es registrar la etiqueta y el manipulador de shortcode gracias a la función de add_shortcode :

```
add_shortcode( ‘shortcode_name’, ‘shortcode_handler’ );
```

La etiqueta de nombre de shortcode es una etiqueta que WordPress buscará en el contenido de la publicación. Debe ser en minúsculas, y sólo requiere letras, números y subrayados
El manipulador de shortcode es una función de devolución de llamada que se ejecutará cuando WordPress encuentre el shortcode

La función del manipulador (handler) se define de la siguiente manera:

```
function shortcode_handler( $atts, $content, $tag ){}
```

La función mantiene tres argumentos:

$atts (array): una matriz de atributos o una cadena vacía;
$content (string): el contenido adjunto (disponible para incluir shortcodes solamente
$tag (string): el nombre del shortcode, útil para funciones de devolución compartida.

WordPress shortcodes pueden ser agregados al archivo o a un plugin functions.php de un tema. En temas podemos ejecutar la función de add_shortcode tal cual, pero en plugins debemos esperar hasta que WordPress haya sido inicializado:

```
function shortcodes_init(){
add_shortcode( 'shortcode_name', 'shortcode_handler' );
}
add_action('init', 'shortcodes_init');
```

Mi Primer Self-closing Shortcode

Nuestro primer shortcode incluye el siguiente elemento en el contenido de la publicación:

```
<a href="" id="" class="" target="">Button</a>
```

Los atributos y el texto serán definidos por los usuarios como sigue:

```
[kinsta_btn href="" id="" class="" target="" label=""]
```

Ahora que hemos establecido nuestro objetivo, vamos a abrir el archivo principal de un plugin y agregar el siguiente código:

```
function kinsta_shortcodes_init(){
```

```
        add_shortcode( 'kinsta_btn', 'kinsta_button' ); }
        add_action('init', 'kinsta_shortcodes_init');
```

Después defina la función de manipulador (handler) de shortcode:

```
function kinsta_button( $atts ){
```

```
   // normalize attribute keys, lowercase
```

```
$atts = array_change_key_case( (array)$atts, CASE_LOWER );

extract( shortcode_atts(
array(
'href' => '',
'id' => '',
'class' => 'green',
'target' => '',
'label' => 'Button'
),
$atts,
'kinsta_btn'
) );

if( in_array( $target, array( '_blank', '_self', '_parent', '_top' ) ) ){
$link_target = ' target="' . esc_attr( $target ) . '"';
}else{
$link_target = '';
}

$output = '<p><a href="' . esc_url( $href ) . '" id="' . esc_attr( $id ) . '"
class="button ' . esc_attr( $class ) . '"' . $link_target . '>' . esc_attr( $label
) . '</a></p>';
return $output;
}
```

Esto es lo que sucede en esta función :

array_change_key_case es una función PHP que devuelve las claves de una matriz en mayúsculas o minúsculas.
shortcode_atts es una función de WordPress que combina los atributos shortcode de usuario con atributos existentes y establece valores predeterminados cuando sea necesario.
La matriz resultante de atributos se pasa a la función de extracto PHP, que importa variables de una matriz a la tabla de símbolos actual.
La condición siguiente comprueba el valor del atributo de destino y establece una cadena vacía si no se encuentra ningún valor válido.
La variable $output almacena el código HTML del elemento de anclaje, que finalmente es devuelto por la función.

Con el fin de convertir el elemento en un botón CSS3, tenemos que registrar y colocar en fila una hoja de estilo a la lista de recursos disponibles:

```
function kinsta_enqueue_scripts() {
global $post;
if( is_a( $post, 'WP_Post' ) && has_shortcode( $post->post_content, 'kinsta_btn') )
{
wp_register_style( 'kinsta-stylesheet', plugin_dir_url( __FILE__ ) .
'css/style.css');
wp_enqueue_style( 'kinsta-stylesheet' );
}
}
add_action( 'wp_enqueue_scripts', 'kinsta_enqueue_scripts');
```

La función define la variable global $post , después verifica dos condiciones:

si $post es una instancia del objeto WP_Post
si el contenido de la publicación contiene el Shortcode kinsta_btn

Si ambas condiciones son verdaderas, la función registra y pone coloca fila la hoja de estilo. No le

mostraré el código CSS aquí, pero puede encontrarlo en Github. Finalmente podemos agregar el shortcode [kinsta_btn] en el contenido de la publicación como sigue:

```
[kinsta_btn href="http://www.google.com" class="blue rounded" target="_blank"]
```

Con el estilo CSS adecuado el ancla se presentará como un increíble botón CSS3

Construyendo un Enclosing Shortcode

Si quisiéramos la misma marca HTML de un enclosing shortcode, haríamos sólo pequeños cambios en la función del manipulador anterior. Primero, registraremos el Shortcode [kinsta_btn_adv]:

```
function kinsta_shortcodes_init(){
```

```
    add_shortcode( 'kinsta_btn_adv', 'kinsta_button_adv' );
    }
    add_action('init', 'kinsta_shortcodes_init');
```

Después, definiremos el nuevo handler:

```
function kinsta_button_adv( $atts, $content = null, $tag = ” ){
```

```
    // normalize attribute keys, lowercase
    $atts = array_change_key_case( (array)$atts, CASE_LOWER );
```

```
extract( shortcode_atts(
array(
'href' => '',
'id' => '',
'class' => 'green',
'target' => ''
),
$atts,
'kinsta_btn'
) );

if( in_array( $target, array( '_blank', '_self', '_parent', '_top' ) ) ){
$link_target = ' target="' . esc_attr( $target ) . '"';
}else{
$link_target = '';
}

$output = '<p><a href="' . esc_url( $href ) . '" id="' . esc_attr( $id ) . '"
class="button ' . esc_attr( $class ) . '"' . $link_target . '>' . esc_attr(
$content) . '</a></p>';
return $output;
}
```

No necesitamos el atributo label para el botón, ya que los usuarios pasarán la cadena a través del contenido shortcode. Teniendo dos shortcodes tenemos que cargar la hoja de estilos bajo dos condiciones. Por lo tanto, vamos a cambiar la función kinsta_enqueue_scripts como sigue:

```
function kinsta_enqueue_scripts() {
```

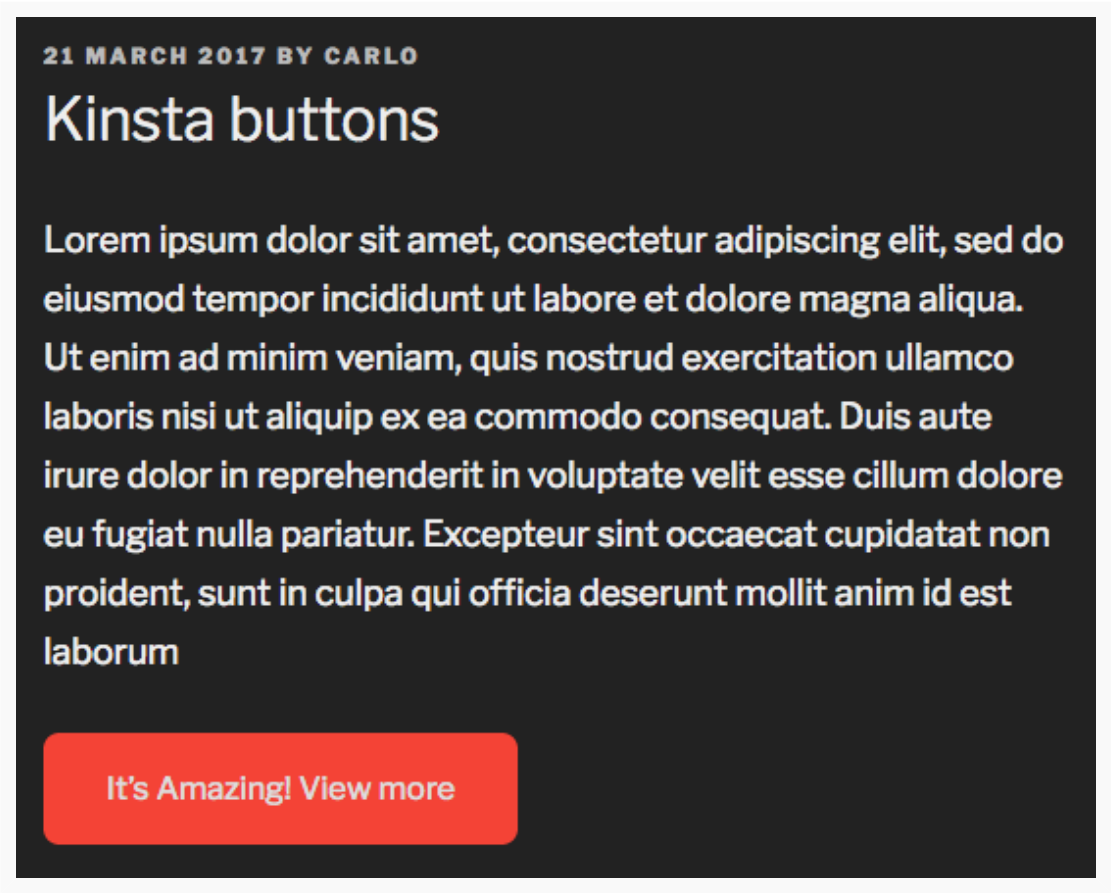
```
global $post;

// see https://codex.wordpress.org/Function_Reference/has_shortcode
$has_shortcode = has_shortcode( $post->post_content, 'kinsta_btn') ||
has_shortcode( $post->post_content, 'kinsta_btn_adv');

if( is_a( $post, 'WP_Post' ) && $has_shortcode ) {
// see https://codex.wordpress.org/Function_Reference/plugin_dir_url
wp_register_style( 'kinsta-stylesheet', plugin_dir_url( __FILE__ ) .
'css/style.css');
// see https://developer.wordpress.org/reference/functions/wp_enqueue_style/
wp_enqueue_style( 'kinsta-stylesheet' );
}
}
add_action( 'wp_enqueue_scripts', 'kinsta_enqueue_scripts');
```

Ahora podemos usar el siguiente enclosing shortcode:

```
[kinsta_btn_adv href=”http://kinsta.com/blog/” class=”red rounded”
target=”_blank”]It’s Amazing! View more[/kinsta_btn_adv]
```



Añadiendo el Shortcode desde Editor Visual

Teclear WordPress shortcodes puede ser bastante molesto y desagradable. Por suerte, podemos hacer esta tarea rápida y divertida gracias al TinyMCE API.

En este último ejemplo, crearemos una interfaz que permitirá a los usuarios insertar shortcodes complejos en el contenido de la publicación con facilidad. Para lograr esta tarea final, necesitamos un plugin de TinyMCE y un botón de MCE. Volvamos a nuestro plugin de WordPress y agreguemos el siguiente código:

```
// register TinyMCE buttons
function kinsta_register_mce_buttons( $buttons ) {
$buttons[] = 'kinsta';
return $buttons;
}
// add new buttons
add_filter( 'mce_buttons', 'kinsta_register_mce_buttons' );
```

Acabamos de agregar un nuevo botón llamado kinsta. Ahora, registremos un plugin de TinyMCE:

```
function kinsta_register_mce_plugin( $plugin_array ) {
```

```
$plugin_array['kinsta'] = plugins_url( '/mce/kinsta/plugin.js', __FILE__ );
return $plugin_array;
}
// Load the TinyMCE plugin
add_filter( 'mce_external_plugins', 'kinsta_register_mce_plugin' );
```

Este código registra el plugin de kinsta TinyMCE, el cual está localizado en la carpeta /mce/kinsta/. Abra el archivo plugin.js y añada el siguiente JavaScript (usted puede ver el archivo en Github):

```
(function() {
```

```
tinymce.PluginManager.add( 'kinsta', function( editor ){});
})();
```

El método PluginManager.add registra el plugin kinsta. Este método nos permite agregar un botón personalizado a la interfaz de usuario del editor:

```
editor.addButton( ‘kinsta’, {
```

```
title: 'Kinsta buttons',
text: 'Kinsta',
icon: 'code',
onclick: function(){}
});
```

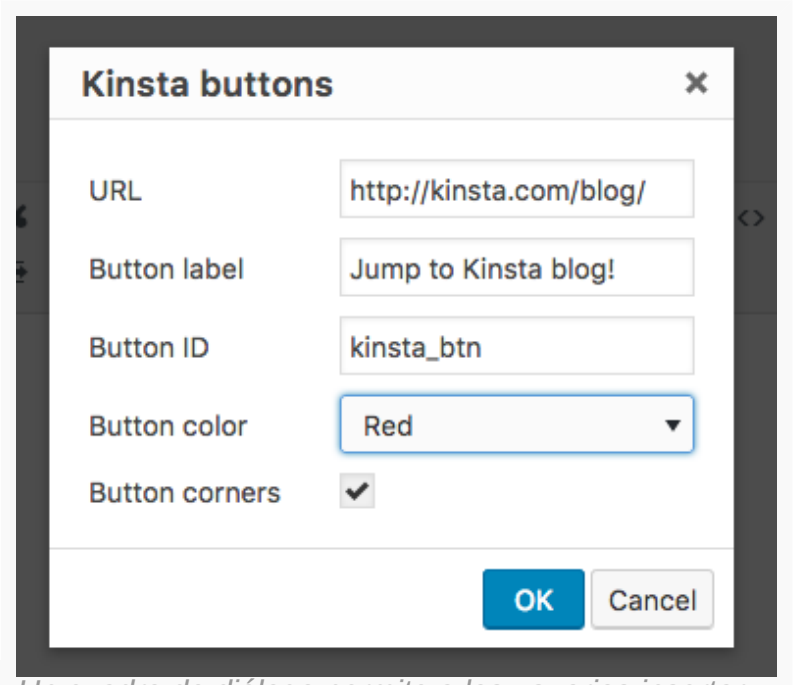
La propiedad onclick define una nueva función anónima, que establece las propiedades de un cuadro de diálogo:

```
onclick: function(){
```

```
editor.windowManager.open({
title: 'Kinsta buttons',
body:
[
{type: 'textbox', name: 'btn_url', label: 'URL'},
{type: 'textbox',name: 'btn_label', label: 'Button label'},
{type: 'textbox', name: 'btn_id', label: 'Button ID'},
{type: 'listbox', name: 'btn_color', label: 'Button color', values: [
{text: 'Green', value: 'green'},
{text: 'Blue', value: 'blue'},
{text: 'Red', value: 'red'},
{text: 'Grey', value: 'grey'},
{text: 'Black', value: 'black'}
]},
{type: 'checkbox', name: 'btn_corners', label: 'Button corners'}
],
onsubmit: function(e){
var tag = 'kinsta_btn';
var href = ' href= "' + e.data.btn_url + '"';
var label = ' label="' + e.data.btn_label + '"';
var id = ' id="' + e.data.btn_id + '"';
var corners = e.data.btn_corners == true ? ' rounded' : '';
var color = e.data.btn_color;
var css_class = ' class="' + color + corners + '"';
editor.insertContent('[' + tag + href + id + css_class + label + ']')
}
})
}
```

El cuadro de diálogo contiene tres cuadros de texto, un cuadro de lista y una casilla de verificación.

Cuando se envía el formulario, el método insertContent incluye el shortcode en el contenido de la publicación.



Un cuadro de diálogo permite a los usuarios insertar shortcodes complejos en el contenido del mensaje con facilidad

En realidad, esto es sólo un ejemplo simple. Podríamos dar a los usuarios más control sobre los WordPress shortcodes, gracias a un buen número de campos de entrada y controles que se enumeran en la documentación de TinyMCE .

Usos No Convencionales de Shortcodes

De inmediato WordPress admite shortcodes sólo en contenido de publicación. De todos modos, como desarrolladores de plugins, podemos superar esta limitación filtrando cualquier contenido de texto en cualquier lugar del sitio. Podemos filtrar widgets de texto:

```
add_filter( ‘widget_text’, ‘shortcode_unautop’);
```

```
add_filter( 'widget_text', 'do_shortcode');
```

La primera línea aplica la función shortcode_unautop al contexto de widgets de texto para evitar que WordPress automáticamente envuelva los shortcodes dentro de los párrafos. La segunda línea aplica la función do_shortcode a widgets de texto.

Podemos filtrar extractos de publicación:

```
add_filter( 'the_excerpt', 'do_shortcode');
```

Y podemos filtrar descripciones de término:

```
add_filter( ‘term_description’, ‘do_shortcode’ );
```

De la misma manera, podemos generar dinámicamente los títulos de los elementos del menú de navegación. Supongamos que usted desea que un elemento de menú muestre el nombre de usuario de la persona que ha iniciado sesión. En primer lugar, necesita un shortcode que muestre el nombre de usuario:

```
function kinsta_shortcodes_init(){
```

```
add_shortcode( 'kinsta_usr', 'kinsta_username' );
}
add_action('init', 'kinsta_shortcodes_init');

function kinsta_username( $atts = array() ){

$id = get_current_user_id();

if ( 0 == $id ) {
// Not logged in
return __( 'Guest' );
} else {
// Logged in
$user = get_userdata( $id );
return $user->user_login;
}
}
```

Ahora puede escribir el contenido de la publicación ‘[kinsta_usr]’ para obtener el login del usuario actual. Para utilizar este shortcode en los elementos del menú, tenemos que filtrar la lista mediante el filtro wp_nav_menu_objects :

```
function kinsta_dynamic_menu_items( $menu_items ) {
```

```
global $shortcode_tags;

foreach ( $menu_items as $menu_item ) {

if ( has_shortcode( $menu_item->title, 'kinsta_usr' ) && isset(
$shortcode_tags['kinsta_usr'] ) ){

$menu_item->title = do_shortcode( $menu_item->title, '[kinsta_usr]' );

if ( 0 == get_current_user_id() ){

$menu_item->url = wp_login_url();

}
}
}
return $menu_items;
}
add_filter( 'wp_nav_menu_objects', 'kinsta_dynamic_menu_items' );
```

Primero, definimos la variable global de $shortcode_tags. Después, marcamos dos condiciones:

si algún elemento del menú contiene el shortcode [kinsta_usr]
si el shortcode [kinsta_usr] existe

Si ambas condiciones son verdaderas, do_Shortcode busca cualquier título de artículo del menu para '[kinsta_usr]' y ejecuta el handler de shortcode. Para usuarios que no han iniciado sesión cambiamos el artículo del menú URL a wp_login_url(). Ahora vaya a Appearance Menus Screen y agregue un nuevo elemento de menú al menú de navegación como se muestra en la imagen de abajo.

Guarde el menú y vea el sitio.

Conclusión

Ahora sabemos cómo construir WordPress shortcodes, cómo permitir que los usuarios los agreguen en cualquier lugar del sitio y cómo construir diálogos de TinyMCE para configurar WordPress shortcodes complejos en poco tiempo.