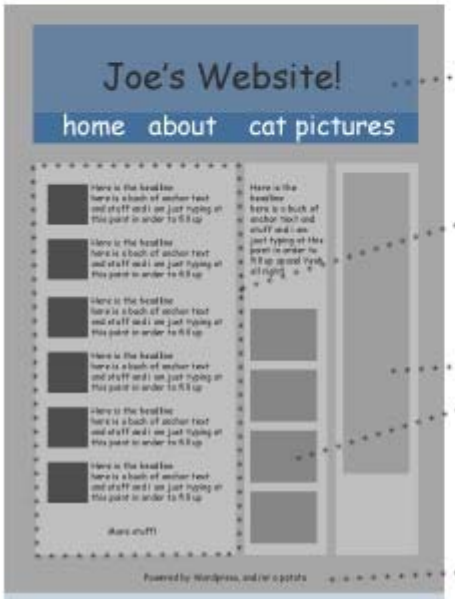
WordPress 2

Anatomía de un Theme de WordPress

Tips para saber cómo funciona tu blog.



header.php

Archivo global que muestra los encabezados y la navegación. También contiene código HTML.

El ciclo

La visualización de contenidos del área principal del sitio son controlados por disitintos archivos del tema de wp.

sidebar.php

Archivo que controla las barras laterales denominadas sidebar y se pueden configurar en functions.php. Los widgets que contienen se instalan desde el wp-admin.

footer.php

Contiene instrucciones para el pie de página global y cierra las etiquetas HTML.

Los temas de WordPress se componen de una carpeta de archivos, cada uno de ellos controla una parte especifica del tema. Partes del sitio que permances estáticos, sin importar lo que se controle por el header, la sidebar o el footer. Tu puedes enganchar esos archivos, los cuales son detectados y que guardan diferente contenido de acuerdo a cada uno de ellos, así como desplegar diferentes tipos de navegación, post o cualquier tipo de información de tu página.

home

index.php

Este archivo controla la lo que aparece en la página principal de WordPress. Por defecto es un ciclo de consultas que muestra las entradas del blog más recientes, con un enlace en la parte inferior para ver los mensajes anteriores.

También se puede especificar en el wp-admin para que aparezca una pagina específica que se haya creado en el panel de administración de WordPress.

posts

single.php

Este pequeño archivo controla y muestra los post individuales de su sitio en WordPress.

Si uno lo desea, se puede especificar un aspecto distinto de las otras páginas del Sitio generadas con WordPress.

pages

page.php

page.php controla las páginas que aparecen en el sitio. Se puede optar por elminar las barras laterales (sidebar) u otros elementos, añadir otros elementos sólo para mostrar en las páginas.
También permite crear diferentes plantillas de páginas dentro de un mismo tema. Para crear una plantilla de página, simplemente se copia este archivo, se renombra y se agrega el siguiente código:

```
<?php
/*
   Template Name: El Nombre de su Tema Aqui
*/
?>
```

archives

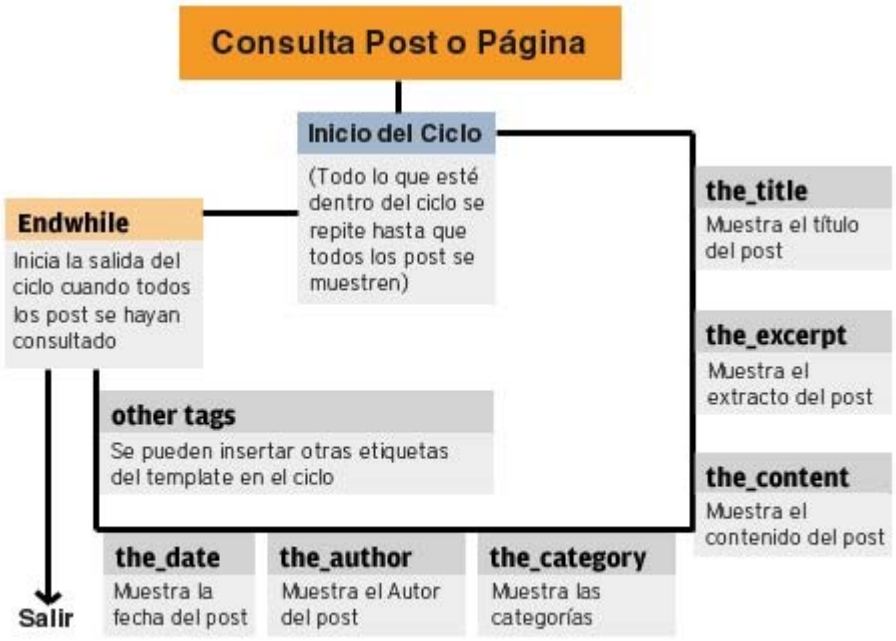
archive.php, category.php, tag.php

También se puede personalizar la apariencia de diferentes archivos de la plantilla como es archive.php, category.php y tag.php. Aunque si no se personalizan creando estos archivos, toman por defecto la apariencia del archivo index.php. Para cambiar la apariencia de cómo se muestra lo archivado del sitio, las categorías y etiquetas, sólo hay que crear dichos archivos (archive.php, category.php y tag.php).

El Ciclo

Es la parte más poderosa del tema de WordPress, se inicia con una consulta (que determina los mensajes y páginas a mostrar) y termina con un end while de PHP. Todo lo demás depende de ti, puedes especificar cómo se muestran los títulos, el contenido de cada post, los metadatos, campos personalizados y de los comentarios.

Se pueden configurar múltimples ciclos y consultas en una misma página.



Detrás de Escena

Para que el template de WordPress trabaje, son necesarios algunos arhivos de fondo. Estos archivos se pueden modificar a sus necesidades y pueden afectar poderosamente la apariencia y funcionalidad del Sitio.

comments.php

Controla la salida de comentarios, que pueden ser incluidos en el ciclo si quiere mostrar comentarios en el sitio. Este archivo se puede reemplazar por plugins que pueden asumir la misma funcionalidad de este archivo.

functions.php

Permite agregar código PHP para modificar los elementos básicos de su tema. Se utiliza para especificar múltiples sidebars, modificar el número de caracteres en el extracto o agregar opciones personalizadas en el panel de administración de wp-admin.

style.css

Es la principal hoja de estilo CSS para el template, también contiene información sobre el tema como es su nombre, el autor y la URL del sitio del autor.

Los Extras

Más allá de la funcionalidad básica de Wordpress, algunos extras como plugins, campos personalizados, y widgets le permiten personalizar aún más su sitio.

Plugins

Una gran comunidad de desarrolladores de código abierto han creado una variedad de plugins que puedes agregar a tu theme de forma gratuita.

Custom Fields

Los campos personalizados le permiten crear sus propias opciones, que se pueden mostrar en el tema. Los usos comunes incluyen imágenes en miniatura, e incluye javascript.

Widgets

Los widgets son pequeños módulos que van dentro de las sidebars. Desde el wp-admin se pueden combinar widgets predefinidos y/o agregar los propios.

La estructura de WordPress

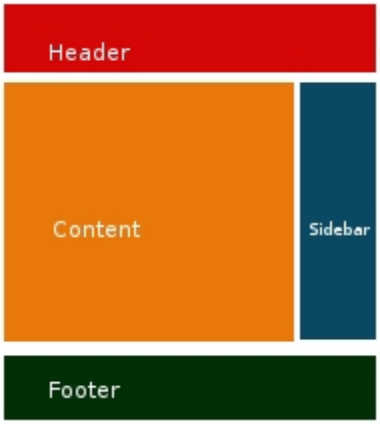
Internamente WordPress se compone de los siguientes ficheros y carpetas:

- **wp-config.php**: fichero de configuración. Aquí está almacenado el usuario y el nombre de la base de datos, así como otros datos de seguridad. Es importante que este fichero no tenga lectura más que por el owner del mismo. Este fichero no se machaca con las actualizaciones de WP.
- **wp-admin** y **wp-includes**: carpetas que guardan los ficheros php que hacen que funcione el blog. Ambas carpetas se machacan con las actualizaciones
- **wp-content**: esta carpeta guarda los datos y configuraciones propias del blog. No se toca nada en la actualizaciones de WP. La estructura interna es la siguiente:
    - **uploads**: todas las imágenes, vídeos y archivos que se suban a la librería de wordpress. Normalmente se organiza por años y luego por meses. Ejemplo: uploads/2009/12.
    - **plugins**: todos los plugins que tenemos disponibles para el blog (aunque no estén activos).
    - **languages**: aquí van los ficheros .mo de los idiomas en los que queremos tener WP (tanto la parte visible como la de administración). Para indicar luego que el blog esté en un idioma u otro, habrá que añadir una línea en wp-config.php como la que sigue: define ('WPLANG', 'es_ES'); donde es_ES es el nombre del fichero .mo (tienen que coincidir en nombre).
    - **themes**: todas las plantillas que tenemos disponibles para "vestir" al blog. Nosotros nos centraremos en esta carpeta.

Como podemos ver, una instalación de WordPress es muy portable. Si nos queremos cambiar de servidor o hosting, bastará con que nos llevemos estas carpetas y la Base de Datos exportada en un simple .sql. No requiere de otras configuraciones más complejas.

La estructura básica de un theme

Las plantillas o themes son la capa de presentación de los blogs o páginas web que se monten. Se puede cambiar esta capa sin que la información se vea afectada. WordPress reconoce automáticamente los siguientes ficheros y los interpreta:



- **header.php**: aquí se define la cabecera de la página.
- **footer.php**: aquí se define el pie de página.
- **sidebar.php**: aquí se definen las diferentes barras laterales.
- **index.php**: este es el fichero que carga de inicio el site. Desde aquí se invoca la carga de cabecera (*get_header();*), barras laterales (*get_sidebar();*) y footer (*get_footer();*) y en el cuerpo central, se ejecuta el bucle para cargar la información de los posts.
- **single.php**: este es el fichero que se interpreta cuando se carga un post concreto.
- **archive.php**: igual que single.php pero para las páginas de categorías y etiquetas.
- **category.php**: plantilla para las categorías. Cuando un usuario pincha sobre el enlace de una categoría, WordPress prueba si existe el fichero *category.php*. Si no lo encuentra, carga *archive.php*. Y si éste tampoco lo encuentra, se va a por el *index.php*. Más adelante veremos un gráfico con la interpretación jerárquica que hace WordPress.
- **comments.php**: plantilla de los comentarios.
- **searchform.php**: formulario de búsqueda.
- **search.php**: página donde se muestran los resultados de una búsqueda.
- **functions.php**: aquí se recogerán funciones propias del theme. Por ejemplo, si queremos registrar varias barras laterales para que aparezcan en la sección de widgets, indicar a WP dónde están los ficheros de idiomas, quitar filtros, …
- **style.css**: fichero con los estilos.
- **404.php**: si creamos este fichero, cada vez que algo no se encuentre en el blog, se cargará esta página.

Desde el apartado de administración se podrá configurar la apariencia de una forma muy visual mediante widgets (en el apartado Apariencia –>Widgets). Aquí aparecen las barras laterales definidas y múltiples funcionalidades que se pueden colocar: comentarios recientes, posts recientes, buscador, una caja donde introducir nuestro propio código html, …

También desde el apartado de administración (Ajustes –> Lectura) podemos configurar que la portada que se carga de nuestra web sea una página y no el formato blog de entradas, dándole más carácter de página web.

Sidebars

Podríamos cargar diferentes sidebars con la misma metodología que los headers y los footers (explicada más adelante), pero existe una manera para que generemos un espacio dentro del apartado de los *widgets*.

Para que aparezcan en la sección de widgets las diferentes sidebars que queramos "alimentar" desde el panel de administración, hay que registrarlas en el fichero functions.php de la siguiente forma:

```
if ( function_exists('register_sidebars') )
{
register_sidebar(array(
'name' => 'Barra Lateral Derecha',
'before_widget' => 'código html que queremos que se cargue al inicio de la barra',
'after_widget' => 'código html que queremos que se cargue al final de la barra',
'before_title' => 'código html que queremos que se cargue al inicio del título de la barra',
'after_title' => 'código html que queremos que se cargue al final del título de la barra',
));
register_sidebar(array(
'name' => 'Barra Lateral Izquierda',
'before_widget' => 'código html que queremos que se cargue al inicio de la barra',
'after_widget' => 'código html que queremos que se cargue al final de la barra',
'before_title' => 'código html que queremos que se cargue al inicio del título de la barra',
'after_title' => 'código html que queremos que se cargue al final del título de la barra',
));
}
```

Para usarlas posteriormente, la llamada será de la siguiente forma:

```
<?phpif ( !function_exists('dynamic_sidebar') || !dynamic_sidebar('Barra Lateral Izquierda') ) :
echo "El usuario no ha puesto widgets en esta barra";
endif; ?>
```

Cabeceras y Footers personalizados

<?php if (is_category('Cine')) {
get_footer('Cine');
} else {
get_footer();
} ?>

De esta forma tan sencilla podemos cargar ficheros php personalizados para mostrar diferentes footers o cabeceras según estemos en una categoría u otra. En el ejemplo, si estamos en la categoría de *Cine* se cargará *footer-cine.php*. Si no, se cargará *footer.php*.

Sería exactamente lo mismo para las cabeceras (*header-cine.php* vs. *header.php*):

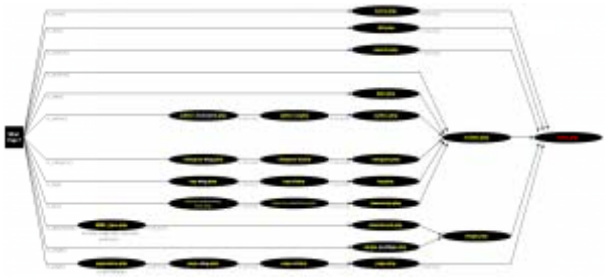
<?php if (is_category('cine')) {
get_header('cine');
} else {
get_header();
} ?>

Es decir, lo que WordPress busca es el fichero *header-slug.php* (el **slug** es el permalink. Es decir, la versión url del nombre: una cadena sin mayúsculas, sin tildes, con guiones en vez de espacios).

Categorías y etiquetas personalizadas

Para cargar diferentes presentaciones dependiendo de la categoría o la etiqueta sobre la que se pinche, podemos crear un php igual que lo hacíamos con las cabeceras, es decir *category-slug.php* o*tag-slug.php* o bien con el siguiente formato de nombre *category-XX.php* o *tag-XX.php* (donde XX es el id que tiene internamente esa categoría o etiqueta). De esta forma, podríamos ponerle una cabecera, un footer y un sidebar personalizados dependiendo de la categoría o la etiqueta en la que estemos.

Aquí vemos el orden en el que WordPress busca los ficheros y los carga:



Para cualquiera de las vistas que no tenga un archivo de plantilla separado, WordPressusará*index.php* de manera predeterminada. Si un visitante solicita un artículo individual, WordPressprimero buscará un archivo llamado *single.php*. Si ese archivo existe, será utilizado para presentar el artículo. Si ese archivo no existe, WordPress utilizará *index.php*.

Templates

Podemos crear plantillas con comportamientos propios a los que luego asignar a cada **página**. Desde el editor de páginas estáticas hay una opción para aplicar plantillas. Para que un fichero php sea interpretado por WordPress como una template, hay que incluir al inicio del mismo el siguiente código:

```
/*
Template Name: Nombre-plantilla
*/
```

The Loop

El bucle es el proceso más importante de WordPress, aquel que nos devuelve y recorre todos los posts que corresponden al fichero desde el que se invoca:

- Si se le llama desde *index.php*, nos devuelve los últimos posts que se hayan escrito (tantos como tengamos definidos que deberían formar parte de la portada en el apartado de administración).
- Si se le llama desde *archive.php*, nos devuelve los posts de una etiqueta determinada o de una categoría concreta (dependerá de quién haga la llamada a ese fichero).

Por tanto, es en este bucle donde procesaremos la colección de posts.

```
<?php
get_header();
if (have_posts()) :
while (have_posts()) : the_post();
the_content();
endwhile;
endif;
get_sidebar();
get_footer();
?>
```

- **wp_query->current_post**: nos devuelve el número del post en el que estamos dentro del loop. Un ejemplo de uso en el que mostramos el contenido para los tres primeros posts y sólo el título para el resto:

```
<?php if (have_posts()) :
while (have_posts()) : the_post();
if ($wp_query->current_post< 3) {
the_content();
}
```

```
else {
the_title();
}
endwhile;
endif; ?>
```

Sin embargo, nosotros también podemos controlar qué posts nos devolverá ese loop mediante la función query_posts. Esa función regenera la consulta y filtra los posts en base a los parámetros que le pongamos:

- **cat=ID**: filtra por esa categoría. Si al id se le pone un guión por delante, muestra los posts de todas las categorías menos de esa.
- **tag=slug**: filtra por etiqueta. En esta ocasión se le pasa el slug. Si queremos usar el ID de la etiqueta, pondremos el parámetro tag_id.
- **author=ID**: filtra por usuario, usando el ID del mismo.
- **order=ASC** u **order=DESC**: indica la ordenación de los resultados, ascendente o descendente.
- **year=año**: filtra por año.
- **monthnum=mes**: filtra por mes.
- **day=día**: filtra por día.
- **posts_per_page=número**: número de posts por página

Ejemplo:

```
<?php
query_posts('posts_per_page=5&author=3&tag=discos');

//The Loop
if ( have_posts() ) : while ( have_posts() ) : the_post();
..
endwhile; else:
..
endif;

//Reset Query
wp_reset_query(); ?>
```

Funciones

- the_permalink(): nos dará la url del post que estemos tratando dentro del loop. No tiene parámetros. Con get_permalink() tendremos que mostrarlo nosotros por pantalla con un echo, pero nos permite que, si tenemos el ID del post y se lo pasamos por parámetro, lo usemos fuera del loop: *get_permalink($post->ID)*.

- the_title($before, $after, $echo): muestra el título del post y también es necesario usarlo dentro del loop. Los parámetros $after y $before nos permite introducir cadenas de texto que irán delante y detrás respectivamente. Por ejemplo, si queremos que el título vaya con las etiquetas html<h3>, podemos hacer la siguiente llamada: the_title('<h3>', '</h3>'). El parámetro $echo nos sirve para indicar si queremos que muestre el título por pantalla (si lo ponemos a *true*) o no para hacer algún tratamiento con ese título (si lo ponemos a *false*). Conget_the_title(ID) sucede igual que get_permalink pero para el título del post.

- the_content(): muestra el cuerpo del post.Se usa dentro del loop. Tiene también su versiónget_the_content().

- the_excerpt(): muestra lo que hayamos introducido en el campo Extracto. Se usa dentro del loop. Tiene su versión get_the_excerpt().

```
<?phpif(!empty($post->post_excerpt)) {
// Si tiene extracto, lo muestra
the_excerpt();
} else {
// Si no, muestra el contenido del post
the_content();
} ?>
```

- the_time(): muestra la fecha por cada post. Le podemos pasar por parámetro el formato de esa fecha:
    - l = Nombre completo del día de la semana.
    - F = Nombre completo para el mes.
    - j = Día numérico.
    - m = Mes con dos dígitos.
    - Y = Año con cuatro dígitos.
    - y = Año con dos dígitos.
    - Para escapar letras, usaremos la barra \. Por ejemplo, para poner la palabra "de" le pasaremos "\d\e"

<?phpthe_time('l, j \d\e F, Y'); //Nos muestra Martes, 14 de Septiembre, 2010 ?>

- the_date(): igual que the_time pero muestra la fecha solo en el primer post de un grupo que haya sido publicado el mismo día.
- Bloginfo: nos ofrece numerosa información de nuestro blog que luego podremos plasmar en otros apartados (tiene se versión get_bloginfo):
    - bloginfo('name'): muestra por pantalla el nombre del blog.
    - bloginfo('description'): muestra por pantalla el nombre del blog.
    - bloginfo('url'): muestra por pantalla la dirección del blog.
    - bloginfo('stylesheet_url'): muestra por pantalla la ruta del fichero de estilos (style.css).
    - bloginfo('template_url'): muestra por pantalla la ruta del theme.
    - bloginfo('rss2_url'): muestra por pantalla la ruta del RSS.

En este ejemplo se muestra el nombre del blog enlazado a su dirección:

<a href="<?phpbloginfo('url'); ?>" title="<?phpbloginfo('name'); ?>"><?phpbloginfo('name'); ?>

- the_author(): muestra por pantalla el autor que ha escrito ese post. Tiene su versión sin echo:get_the_author.
- the_tags($before, $separator, $after): muestra las etiquetas asociadas a ese post. Por parámetro se le puede pasar el texto que precederá a las etiquetas, los caracteres que queremos que separen las etiquetas y el texto que irá al final. Tiene también su versiónget_the_tags que nos devolverá un array con las etiquetas, para que las procesemos nosotros.

*<?phpthe_tags('Etiquetas:', ' – ', '
'); ?>*

- posts_nav_link: muestra un enlace con el texto que le pasemos por parámetro a los posts que estén en páginas anteriores. Se usa en el index.php para que se pueda navegar a los contenidos anteriores a los posts que se muestran en la portada.
- previous_posts_link: muestra un enlace con el texto que le pasemos por parámetro a los posts que estén en páginas anteriores. Se usa en el index.php para que se pueda navegar a los contenidos anteriores a los posts que se muestran en la portada.

- next_posts_link: muestra un enlace con el texto que le pasemos por parámetro a los posts que estén en páginas posteriores. Se usa en el index.php para que se pueda navegar a los contenidos posteriores a los posts que se muestran en la página desde la que se invoca.

```
<div>
<div><?phpprevious_posts_link(”,”,’&laquo; Artículos Anteriores’)?></div>
<div><?phpnext_posts_link(”,’Artículos Siguientes &raquo;’,”)?></div>
</div>
```

- in_category($category): nos dice si ese post está en la categoría pasada por parámetro. Se puede pasar el ID de la categoría o bien el slug. Con esto también podemos hacer que tenga comportamientos distintos según la categoría y en un único fichero category.php.

```
<?php if (in_category(’3’) ){ ?>
<imgsrc=’/images/plant.png’ alt=’a plant’ />
<?php } elseif (in_category(’4’) ){ ?>
<imgsrc=’/images/flower.png’ alt=’a pretty flower’ />
<?php } ?>
```

Customfields – Campos personalizados

Cuando estamos creando un post, justo debajo de la caja del texto podemos introducir campos personalizados introduciendo un nombre y un valor:

El **Nombre** será el identificador de nuestro campo personalizado y el **Valor** lo que queremos mostrar. Obtendremos el valor de ese campo dentro del loop con la siguiente llamadaget_post_custom_values:

```
<?php $valor_custom_field = get_post_custom_values(“nombre-del-custom-field”); ?>
```

O también con la función get_post_meta:

```
<?php $valor_custom_field = get_post_meta($post->ID, nombre-del-custom-field); ?>
```

Estos customfields son los metadatos extra de cada post (que se suman a los ya típicos autor, fecha, etiquetas, categorías, etc…). Así podría ser la canción que está escuchando el autor cuando escribió el post o su estado de ánimo, el precio (si usamos el blog como tienda virtual), etc…

Panel de administración del theme

Para crear un apartado de administración de nuestro theme, debemos incluir la siguiente llamada en functions.php (en este ejemplo estamos poniendo un textarea para que se almacene nuestro código de Google Analytics):

```
/* Meter código de Google Analytics */
function nombredeltheme_theme() {
if(isset($_POST['submitted']) and $_POST['submitted'] == 'yes') :
update_option("google_analytics", stripslashes($_POST['google_analytics']));
endif;
?>
<form method="post" name="update_form" target="_self">
<h1>Google Analytics</h1>
<table>
<tr>
<th>Google Analytics:</th>
<td><textarea name="google_analytics" style="width: 95%;" rows="10" />
<?php echo get_option('google_analytics'); ?></textarea><br />Copia el código Google Analytics aquí.</td>
</tr>
</table>
<p id="jump_submit">
<input name="submitted" type="hidden" value="yes" />
<input type="submit" name="Submit" value="Save Changes" />
</form>
<?php
}
function nombredeltheme_options() {
add_menu_page('Opciones del Theme', __('Opciones del Theme', 'default'), 'edit_themes', __FILE__, 'nombredeltheme_theme');
}
add_action('admin_menu', 'nombredeltheme_options');
```

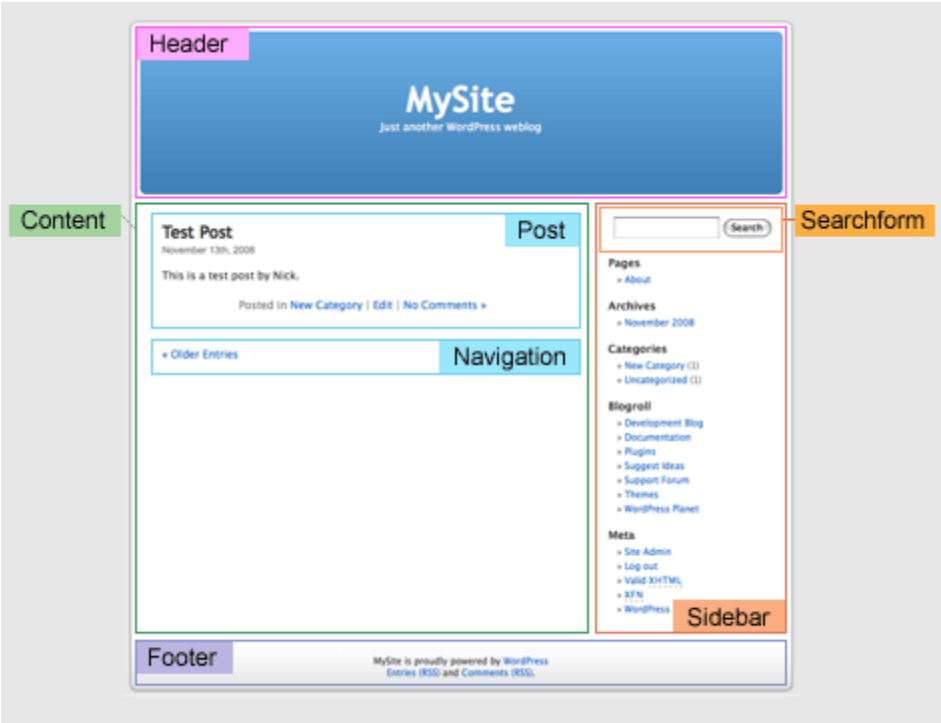
Luego ponemos en el footer.php:

```
<?php echo get_option('google_analytics'); ?>
```

De esta forma se cargará en el footer lo que el usuario alimente desde el panel de administración.

El Frontend del blog

Antes de empezar, vamos a echar un vistazo en el tema por defecto de WordPress y ver cómo está estructurada. Tome nota de los elementos (encabezado, título de la entrada, el formulario de búsqueda, navegación, pie de página, etc.)



Default Frontpage (index.php)

Default Single (single.php)

2. Photoshop Mockups

Con base en la información obtenida a partir del tema por defecto, diseñar una maqueta de Photoshop de tu blog. Aquí estoy usando GlossyBlue, uno de mis temas para WordPress gratuitos, como un ejemplo. Descargue el demo.zip para ver el archivo de Photoshop.

Homepage

Single

3. HTML + CSS

Después de que el diseño de PSD se hace, crear una plantilla HTML + CSS estática de cada página. Puedes usar mis archivos HTML GlossyBlue en el demo.zip seguir este tutorial. Extrae el archivo zip y echar un vistazo en el index.html, single.html y page.html. Más adelante en el tutorial, voy a utilizar estos archivos HTML y convertirlos en un tema.

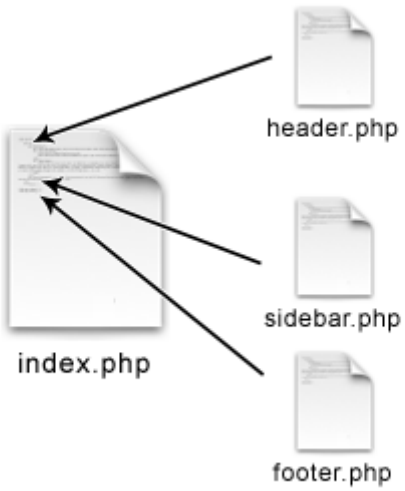
¿Por qué crear un archivo HTML estático en primer lugar?

Principalmente porque hará que el proceso de desarrollo mucho más fácil. Generalmente creo un archivo HTML para cada plantilla que necesito, probarlo en todos los navegadores, validar HTML y CSS márgenes, a continuación, todo lo que tengo que hacer es cortar y pegar el código de WordPress. De esta manera, no tiene que preocuparse acerca de los errores de HTML o CSS en mi tema de decisiones.

4. ¿Cómo funciona un theme de wordpress?

Si vas a la carpeta del tema por defecto (wp-content/themes/default), debería ver muchos archivos PHP (llamado archivo de plantilla) y un archivo style.css. Si va a ver la primera página,

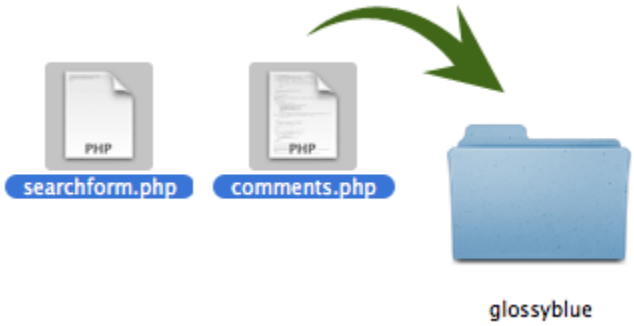
WordPress realmente utiliza varios archivos de plantilla para generar la página (index.php<<header.php, sidebar.php y footer.php).



Para más información, echa un vistazo a la arquitectura del sitio y la Jerarquía Plantilla en el Codex.

5. Duplicar los archivos de plantilla

Copia la carpeta HTML GlossyBlue en la carpeta wp-content/themes. Luego, vaya a la carpeta del tema por defecto, copia el comments.php y searchform.php archivo a la carpeta glossyblue.



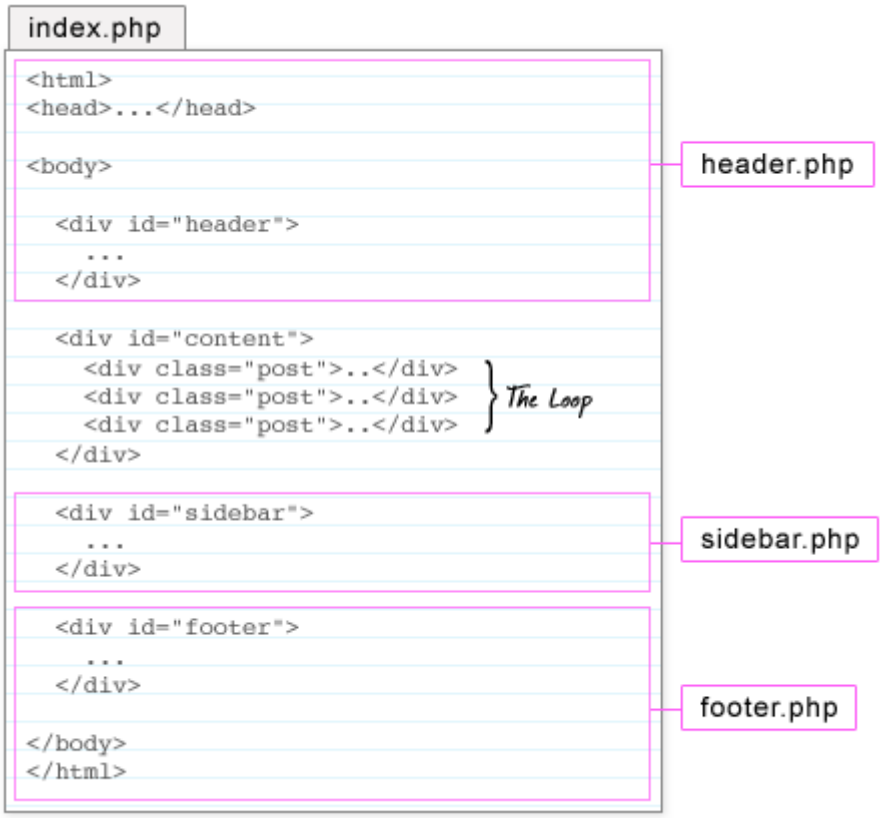
6. style.css

Ir a la carpeta del tema por defecto de WordPress, abra el archivo style.css. Copie el código comentado en la parte superior y pegarlo en el archivo style.css GlossyBlue. Cambie el nombre del tema y la información del autor como usted desea.

```
/*
Theme Name: GlossyBlue
Theme URI: http://www.ndesign-studio.com
Description: A theme by <a href="http://www.ndesign-studio.com">N.Desi
Version: 1
Author: Nick La
Author URI: http://www.ndesign-studio.com

*/
```

7. Dividir Archivos

Ahora lo que necesita para entender dónde dividir el archivo en varios archivos: header.php, sidebar.php y footer.php. La siguiente imagen muestra una versión simplificada de mi archivo de índice y cómo las marcas de revisión deben dividir.



8. Header.php

Abra el archivo index.html. Cortar la parte superior para que el <--/header -> extremos, péguelo en un nuevo archivo PHP y guarde el archivo como header.php.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http:
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
<title>index.html</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="page">
   <div id="header">
      <div id="headerimg">
         <h1><a href="#">My Blog</a></h1>
         <div class="description">Blog Description</div>
      </div>
      <ul id="nav">
         <li><a href="#">Home</a></li>
         <li><a href="#">About</a></li>
         <li><a href="#">Portfolio</a></li>
         <li><a href="#">Links</a></li>
         <li><a href="#">Contact</a></li>
      </ul>
   </div>
   <!--/header -->
```

Vaya a la carpeta predeterminada de temas, abrir las header.php. Copiar y reemplazar las etiquetas donde se requiere que el código PHP (Tag plantilla): <title>, <link> hoja de estilo, <h1> y .

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title><?php bloginfo('name'); ?> <?php if ( is_single() ) { ?> &raquo; Blog Archive <?php } ?> <?php wp_title(); ?></title>

<link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />

</head>
<body>
<div id="page">
   <div id="header">
      <div id="headerimg">
         <h1><a href="<?php echo get_option('home'); ?>/"><?php bloginfo('name'); ?></a></h1>
         <div class="description"><?php bloginfo('description'); ?></div>
      </div>
      <ul id="nav">
```

Navegación por los menús (wp_list_pages)

Reemplace las etiquetas <li> del id=nav><ul con <wp_list_pagesphp ('sort_column = menu_order y profundidad = 1 &title_li =');?>

```
<div id="header">
   <div id="headerimg">
      <h1><a href="<?php echo get_option('home'); ?>/"><?php bloginfo('nc
      <div class="description"><?php bloginfo('description'); ?></div>
   </div>
   <ul id="nav">
      <?php wp_list_pages('sort_column=menu_order&depth=1&title_li=');?>
   </ul>
</div>
<!--/header -->
```

9. Sidebar.php

Volver al archivo index.html, corte desde donde el id=searchform><form principio a la etiqueta de cierre y péguelo en un nuevo archivo PHP, guárdelo como sidebar.php.

- Replace the `<form id=searchform>` wrap with `<?php include (TEMPLATEPATH . '/searchform.php'); ?>`.
- Replace the category `<li>` tags with `<?phpwp_list_categories('show_count=1&title_li='); ?>`
- Replace the archive `<li>` tags with `<?phpwp_get_archives('type=monthly'); ?>`

```
1   <?php include (TEMPLATEPATH . '/searchform.php'); ?>
2
3   <div id="sidebar">
4      <h3>Category</h3>
5      <ul class="ul-cat">
6         <?php wp_list_categories('show_count=1&title_li='); ?>
7      </ul>
8      <h3>Archives</h3>
9      <ul class="ul-archives">
10        <?php wp_get_archives('type=monthly'); ?>
11     </ul>
12  </div>
13  <!--/sidebar -->
14
```

10. Footer.php

Volver al archivo index.html, corte de la etiqueta id=footer><div al final de </ html> y péguelo en un nuevo archivo PHP, guárdelo como footer.php.

```
   <div id="footer">
      <div class="left-col">
         <h4>Recent Posts </h4>
         <ul class="recent-posts">
            <li><a href="#">Blog Title Text</a><br />
               10/23/2006</li>
            <li><a href="#">Blog Title</a><br />
               10/23/2006</li>
            <li><a href="#">Title Text</a><br />
               10/23/2006</li>
            <li><a href="#">Sample Blog Title</a><br />
               10/23/2006</li>
         </ul>
      </div>
      <div class="left-col">
         <h4>Recently Commented</h4>
         <ul class="recent-comments">
            <li><a href="#">Nick:</a> This is a test comment</li>
            <li><a href="#">Jen:</a> Hello, nice day!</li>
            <li><a href="#">Someone:</a> Wow, this is another test comm
            <li><a href="#">Admin:</a> This is a comment</li>
         </ul>
      </div>
      <div class="right-col">
         <h4>About</h4>
         <p>Welcome to my design blog and portfolio showcase. Duis aute
facilisis.</p>
         <p>Feel free to <a href="#"> contact me.</a></p>
      </div>
      <hr class="clear" />
   </div>
   <!--/footer -->
</div>
<!--/page -->
<div id="credits"><span class="left">Powered by <a href="http://www.wor
class="right"><a href="#" class="rss">Entries RSS</a> <a href="#" class
</body>
</html>
```

Entradas Recientes

Aquí he utilizado la query_post para mostrar las 5 últimas entradas.

```
<div id="footer">
   <div class="left-col">
      <h4>Recent Posts </h4>

      <?php query_posts('showposts=5'); ?>
      <ul class="recent-posts">
      <?php while (have_posts()) : the_post(); ?>
         <li>
            <strong><a href="<?php the_permalink() ?>" rel="bookmark" title
            <small><?php the_time('m-d-Y') ?></small>
         </li>
      <?php endwhile;?>
      </ul>

   </div>
```

Comentarios Recientes

Comentarios recientes son generados por un plugin (incluida en la carpeta del tema). Aquí utilicé la query_post para mostrar las 5 últimas entradas.

```
<div class="left-col">
   <?php include (TEMPLATEPATH . '/simple_recent_comments.php'); /* recent c
   <?php if (function_exists('src_simple_recent_comments')) { src_simple_rec
</div>
<div class="right-col">
   <h4>About</h4>
   <p>Welcome to my design blog and portfolio showcase. Duis autem vel eum ir
sis.</p>
   <p>Feel free to <a href="#"> contact me.</a></p>
</div>
```

Plugin

11. Index.php

Ahora en su archivo index.html, sólo debe tener la id=content> envoltura <div. Guarde el archivo como index.php. Inserte la línea: get_header, get_sidebar y get_footer en el mismo orden que la estructura de su diseño.

```
1  <?php get_header(); ?>
2
3     <div id="content">
4        <div class="post">
5           <div class="post-date"><span class="post-month">Oct</span> <s
6           <div class="post-title">
7              <h2><a href="#">Sample Blog Entry</a></h2>
8              <span class="post-cat"><a href="#">News</a></span> <span c
9           </div>
10          <div class="entry">
11             <p>Duis autem vel eum iriure dolor in hendrerit in vulputa
   odio dignissim qui blandit praesent luptatum zzril delenit augue duis
   euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. <a hr
12          </div>
13       </div>
14       <div class="navigation"> <span class="previous-entries"><a href=
15    </div>
16    <!--/content -->
17
18 <?php get_sidebar(); ?>
19 <?php get_footer(); ?>
```

12. Entendiendo elLoop de Wordpress

La imagen de abajo muestra cómo funciona elLoop. El bucle se utiliza para mostrar mensajes de blog y también le permite controlar lo que desea mostrar. Básicamente, TheLoop comprueba si hay mensajes en su blog, mientras que hay puestos de trabajo, muestran que, si hay post encontrado, por ejemplo "NotFound".

13. Copiar elLoop

Ir a la carpeta del tema por defecto, abra el archivo index.php. Copie el lazo de la index.php por defecto y pegarlo en entre el ..</ div>. A continuación, reemplace el texto estático con las plantillas de WordPress Etiquetas: fecha, título, categoría, comentarios, siguiente y anterior enlace.

14. Ver el Tema

¡Enhorabuena! Usted ha hecho la primera página (la parte principal del tema). Ahora, ingrese a su panel de administración, vaya a la pestaña Diseño, debería ver el tema GlossyBlue, activarla y vaya a la primera página para obtener una vista previa del tema.

15. Single.php

Ahora, es el momento de hacer la plantilla single.php. Si lo desea, puede pasar por el mismo proceso - cortar y pegar desde el tema por defecto. Sin embargo, me resulta más fácil utilizar el index.php que acaba de crear y guárdelo como single.php. Abra el archivo single.php tema por defecto y copiar las plantillas Etiquetas más. A continuación, incluya el comments_template. La imagen de abajo destaca lo he cambiado:

```php
<?php get_header(); ?>

   <div id="content">

   <?php if (have_posts()) : ?>

      <?php while (have_posts()) : the_post(); ?>

      <div class="post">
         <div class="post-date"><span class="post-month"><?php the_time
         <div class="post-title">
            <h2><?php the_title(); ?></h2>
            <span class="post-cat"><?php the_category(', ') ?></span>
         </div>
         <div class="entry">

            <?php the_content('Read the rest of this entry &raquo;'); ?

            <?php wp_link_pages(array('before' => '<p><strong>Pages:</s

         </div>

         <?php comments_template(); ?>

      </div>

      <?php endwhile; ?>

      <div class="navigation">
         <span class="previous-entries"><?php previous_post_link(' %lin
         <span class="next-entries"><?php next_post_link('%link') ?></s
      </div>

   <?php else : ?>

      <h2>Not Found</h2>
      <p>Sorry, but you are looking for something that isn't here.</p>

   <?php endif; ?>

   </div>
   <!--/content -->

<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

16. Page.php

Con la plantilla single.php que acaba de crear, guardarla como page.php. Retire la fecha, forma de comentario, siguiente / anterior enlace ... y eso es todo ..ahí va la plantilla page.php.

17. Elimine los archivos HTML

Elimine todos los archivos HTML en la carpeta glossyblue (no los necesitamos más). Técnicamente, eso es suficiente para un tema básico de WordPress. Usted puede notar que hay más archivos PHP en el tema por defecto. Bueno, realmente no necesita los archivos si lo que desea es un tema básico. Por ejemplo, si el search.php o 404.php no está en la carpeta del tema, WordPress usará automáticamente el index.php para representar la página. Leer la Jerarquía Plantilla para más detalles.

18. Plantilla de PáginaWordpress

Ok, último ejemplo. Yo te mostraré cómo usar plantillas de página para crear una página de archivo que lista todos los mensajes en tu blog (bueno para sitemap). Copie el archivo

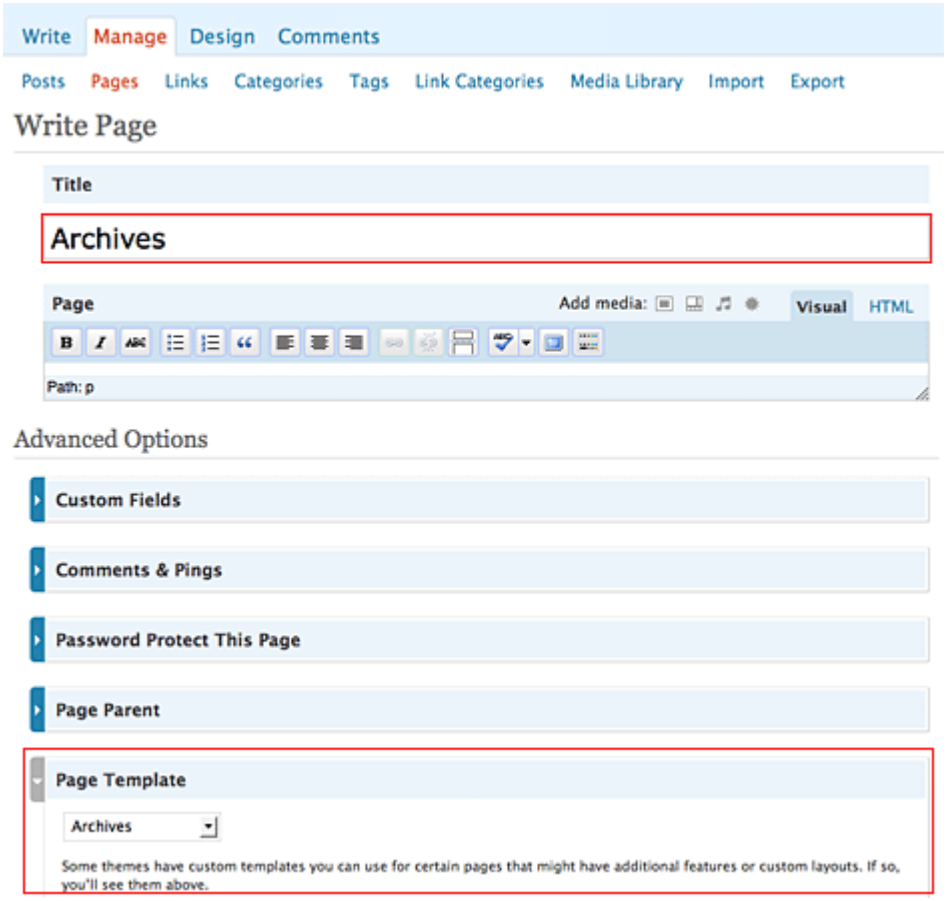
archives.php de la carpeta predeterminada de temas. Elimine el código deseado y usted debe tener algo como esto:

```php
<?php
/*
Template Name: Archives
*/
?>

<?php get_header(); ?>

<div id="content">

   <h2><?php the_title(); ?></h2>



</div>

<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

Aquí estoy usando el query_post (showposts = -1 significa mostrar todas las entradas) para mostrar una lista de todos los mensajes.

```php
<?php
/*
Template Name: Archives
*/
?>

<?php get_header(); ?>

<div id="content">

   <h2><?php the_title(); ?></h2>

   <?php query_posts('showposts=-1'); ?>

   <ul>
      <?php while (have_posts()) : the_post(); ?>
      <li>
         <h3><a href="<?php the_permalink() ?>" title="<?php the_title
         <?php the_time('M d Y') ?> | <?php comments_popup_link('No C
      </li>
      <?php endwhile;?>
   </ul>

</div>

<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

Ahora, ingrese a su panel de administración, escribir una nueva página, título que Archives. En la plantilla de página desplegable, seleccione Archivos.



Crear un Widget para WordPress

Muchas veces nos gustaría mostrar cierta información en nuestra barra lateral (o Sidebar) de WordPress, y recurrimos a añadir un Widget de texto plano con HTML. Esto está bien, es simple y cómodo, pero sólo para texto estático. En el momento en que queremos algo más dinámico, no podemos (pues no podemos insertar código PHP). Es por ello que lo ideal es crear un Widget para WordPress con el contenido que queramos.

Vamos a empezar con un ejemplo simple, y lo iremos complicando poco a poco.

Creando el Widget

Un Widget se compone de dos partes:

- Una función que instancia al Widget.
- Una clase para nuestro Widget, que extiende al objeto WP_Widget.

Ya hemos explicado en más de una ocasión que lo ideal es crear un plugin para nuestro Widget, pero debo decir que sí, que también podéis añadir la función que instancia el Widget en el archivo functions.php. Nosotros crearemos un nuevo plugin, cuyo código es simplemente este:

```
<?php

/*
Plugin Name: Mi primer Widget
Plugin URI: http://www.web-sistemas.com/
Description: Crea un Widget para añadir a cualquier Sidebar.
Version: 1.0
Author: web-sistemas
Author URI: http://www.web-sistemas.com/
*/

/**
 * Función que instancia el Widget
 */
function mpw_create_widget(){
   include_once(plugin_dir_path( __FILE__ ).'/includes/widget.php');
   register_widget('mpw_widget');
}
add_action('widgets_init','mpw_create_widget');

?>
```

Vale, ¿Esto qué hace?

Tras declarar la información del plugin, tenemos la función mpw_create_widget (mpw viene de "Mi Primer Widget", el nombre de la función podéis cambiarlo como queráis, claro). Esta función hace dos cosas:

1. Incluir el archivo widget.php, que crearemos en la carpeta wp-content/<nombre_de_tu_plugin>/includes/widget.php. Esta es mi forma de tener los archivos organizados, quizá vosotros prefiráis no usar la carpeta includes, o usar /includes/widget/widget.php. En nuestro caso, lo haremos así.
2. Registrar el widget en el sistema con la función "register_widget". Como parámetro, recibe el nombre que le daremos a la clase del Widget que vamos a crear, en este caso, mpw_widget.

La clase de nuestro Widget

Si no hemos creado el archivo widget.php, es el momento de hacerlo. Nuestra clase debe extender a la clase WP_Widget, y debe tener cuatro funciones básicas:

```
<?php
```

```
class mpw_widget extends WP_Widget {

   function mpw_widget(){
      // Constructor del Widget.
   }

   function widget($args,$instance){
      // Contenido del Widget que se mostrará en la Sidebar
   }

   function update($new_instance, $old_instance){
      // Función de guardado de opciones
   }

   function form($instance){
      // Formulario de opciones del Widget, que aparece cuando añadimos el Widget a una Sidebar
   }
}

?>
```

Esta es una plantilla válida para cualquier Widget que os planteéis crear. Ahora, detengámonos a explicarlas una a una.

La primera función debe llamarse igual que la clase. Es la función constructora, la que se instancia una vez que el Widget es llamado. En esta función, lo básico que debemos hacer es llamar al constructor de WP_Widget con las opciones del Widget:

```
 function mpw_widget(){
      $widget_ops = array('classname' => 'mpw_widget', 'description' => "Descripción de Mi primer Widget" );
      $this->WP_Widget('mpw_widget', "Mi primer Widget", $widget_ops);
   }
```

Esto hará que nuestro Widget sea visible en la zona de Widgets de WordPress.

La siguiente función es “widget”. Esta función es la que genera el contenido que se muestra en la zona del Widget, lo que verán tus usuarios en el Front End. Luego lo complicaremos, pero de momento, veamos un ejemplo simple:

```
function widget($args,$instance){
      echo $before_widget;
      ?>
      <aside id='mpw_widget' class='widget mpw_widget'>
         <h3 class='widget-title'>Mi Primer Widget</h3>
         <p> ¡Este es mi primer Widget!</p>
      </aside>
      <?php
```

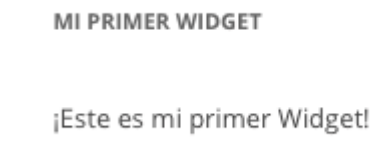
```
    echo $after_widget;
}
```

Activando el Widget

Llegados a este punto, supongo que habéis activado el plugin desde el gestor de plugins de WordPress. Hecho esto, vamos a Apariencia > Widgets, y encontraremos nuestro Widget en la lista de Widgets disponibles. Tan solo tenemos que arrastrarlo a la Sidebar que mejor nos convenga.

Si no ves tu Widget en la lista, algo has pasado por alto. Revisa las instrucciones hasta aquí.

Finalmente, vamos al Front End y vemos el resultado.



Complicando el Widget con configuración interna

Si nuestro Widget está hecho sólo para mostrar información concreta, y no requiere de configuración interna, habríamos acabado aquí, no serían necesarias más funciones. En el caso de que requiera configuración, entonces debemos crear las funciones update y form.

```
function update($new_instance, $old_instance){
      $instance = $old_instance;
      $instance["mpw_texto"] = strip_tags($new_instance["mpw_texto"]);
      // Repetimos esto para tantos campos como tengamos en el formulario.
      return $instance;
   }
```

La función update se encarga de guardar en la base de datos la configuración establecida para el Widget. Suele seguir una estructura similar a la que vemos siempre, cambiando los parámetros de los campos.

La función form es la que muestra el formulario de configuración del Widget en el Back End de WordPress. Por ejemplo, vamos a mostrar un texto:

```
function form($instance){
      ?>
       <p>
         <label for="<?php echo $this->get_field_id('mpw_texto'); ?>">Texto del Widget</label>
         <input class="widefat" id="<?php echo $this->get_field_id('mpw_texto'); ?>" name="<?php echo $this-
>get_field_name('mpw_texto'); ?>" type="text" value="<?php echo esc_attr($instance["mpw_texto"]); ?>" />
       </p>
       <?php
   }
```

Las funciones get_field_id y get_field_name las usamos para que el guardado del Widget sea correcto y coherente en cuanto a parámetros.

Ahora debemos adaptar nuestra función widget para que acepte y muestre este texto:

```
function widget($args,$instance){
    echo $before_widget;
    ?>
    <aside id='mpw_widget' class='widget myp_widget'>
        <h3 class='widget-title'>Mi Primer Widget</h3>
        <p><?=$instance["mpw_texto"]?></p>
    </aside>
    <?php
    echo $after_widget;
}
```

Y podemos comprobar que, efectivamente, aparece el texto que hemos colocado:

Creando nuevos widgets y areas para widgets

Para crear nuevos widgets y ponerlos en la posición de nuestro tema en wordpress primero que nada debemos de crear el archivo como se llame nuestro widget

Ejemplo: publicidad-arriba.php

Y el código que este llevara ya que mis necesidades por ahora es solo colocar nuevo código html, para colocar imágenes de publicidad o banners en flash entonces por eso utilizo este código y solo acepto código html y sin título para que no aparezca nada en dicha posición más que el código de la publicidad:

y lo debemos guardar en el directorio de widgets de nuestro theme ejemplo:

includes\widgets\publicidad-arriba.php y el código de dicho widget es el siguiente:

```
*widget realizado por Geovanny Quilarque*/
/*visita para mas información www.web-sistemas.com*/

<?php class PublicidadArribaWidget extends WP_Widget
{
   function PublicidadArribaWidget(){
                        $widget_ops = array( 'description' => 'Para colocar la publicidad arriba' );
                        $control_ops = array( 'width' => 400, 'height' => 300 );
                        parent::WP_Widget( false, $name='Publicidad Arriba Widget', $widget_ops,
$control_ops );
   }
/* Displays the Widget in the front-end */
```

```
   function widget( $args, $instance ){
                        extract($args);
                        $aboutText = empty( $instance['aboutText'] ) ? '' : $instance['aboutText'];
                        echo $before_widget;
      echo( $aboutText );?>
            <?php
                        echo $after_widget;
            }
    /*Saves the settings. */
   function update( $new_instance, $old_instance ){
                        $instance = $old_instance;
                        $instance['title'] = strip_tags( $new_instance['title'] );
                        $instance['imagePath'] = esc_url( $new_instance['imagePath'] );
                        $instance['aboutText'] = current_user_can('unfiltered_html') ?
$new_instance['aboutText'] : stripslashes( wp_filter_post_kses( addslashes($new_instance['aboutText']) )
);
                        return $instance;
            }


    /*Creates the form for the widget in the back-end. */
   function form( $instance ){
                        //Defaults
                        $instance = wp_parse_args( (array) $instance, array( 'title'=>'About Me',
'imagePath'=>'', 'aboutText'=>'' ) );
                        $title = esc_attr( $instance['title'] );
                        $imagePath = esc_url( $instance['imagePath'] );
                        $aboutText = esc_textarea( $instance['aboutText'] );
                        # Title
                        echo '<p><label for="' . $this->get_field_id('title') . '">' . 'Title:' . '</label><input id="'
. $this->get_field_id('title') . '" name="' . $this->get_field_name('title') . '" type="text" value="' . $title .
'" /></p>';
                        # Image
                        echo '<p><label for="' . $this->get_field_id('imagePath') . '">' . 'Image:' .
'</label><textarea cols="20" rows="2" id="' . $this->get_field_id('imagePath') . '" name="' . $this-
>get_field_name('imagePath') . '" >'. $imagePath .'</textarea></p>';
                        # About Text
                        echo '<p><label for="' . $this->get_field_id('aboutText') . '">' . 'Text:' .
'</label><textarea cols="20" rows="5" id="' . $this->get_field_id('aboutText') . '" name="' . $this-
>get_field_name('aboutText') . '" >'. $aboutText .'</textarea></p>';
            }
}// fin del widget de publicidad arriba class


function PublicidadArribaWidgetInit() {
            register_widget('PublicidadArribaWidget');
}


add_action('widgets_init', 'PublicidadArribaWidgetInit'); ?>
```

En caso de que ustedes quieran crear un widgets con funciones más avanzadas pueden basarse en otros widgets ya escritos y personalizarlos a su gusto o buscar directamente en la página de codex de wordpress para ver más ejemplos y todos los parámetros que pueden utilizar para sus widgets

Y para que pueda aparecer en nuestro tema en el back end para primero visualizar que ya esta disponible entonces lo primero que hacemos es agregarlo también en el archivo widgets.php del theme, en algunos casos en el archivo functions.php

Y quedaría de la siguiente forma el archivo widgets.php final

```
<?php get_template_part('includes/widgets/widget-about');
get_template_part('includes/widgets/widget-adsense');
get_template_part('includes/widgets/widget-ads');
get_template_part('includes/widgets/widget-fromblog');
get_template_part('includes/widgets/widget-recentvideos');
get_template_part('includes/widgets/widget-photostream');
get_template_part('includes/widgets/widget-popular');
get_template_part('includes/widgets/widget-728');
get_template_part('includes/widgets/publicidad-arriba');
?>
```

Después si refrescamos en el área de widgets podremos visualizar que podemos utilizar nuestro nuevo widget para poder colocarlos en el sidebar o en una de las áreas de widgets que tiene nuestro tema.

Ahora bien como en mi caso no quiero que aparezca la publicidad en el sidebar si no que quiero que aparezca en la parte superior antes del menú superior (El primero) ya que este tema cuenta con tres, entonces ahora lo que debemos de hacer es crear una nueva área para widgets y plugins.

Para crear una nueva área para widgets y plugins dentro de nuestro tema yo lo voy a declarar en el archivo sidebars.php en mi caso ubicado en: \wp-content\themes\Aggregate\includes\functions\ sidebars.php

En donde se declaran todas las áreas para widgets y plugins en mi caso voy a duplicar uno existente y lo voy a modificar para que quede de la siguiente manera:

```
register_sidebar( array(
                'name' => 'Publicidad Arriba',
                'id' => 'PublicidadArriba',
                'before_widget' => '',
                'after_widget' => '',
                'before_title' => '',
                'after_title' => '',
        ) );
```

Si queremos que aparezca antes de todos los widgets nos aseguramos de que sea el primer widget declarado en donde empieza el archivo.

```
<?php
if ( function_exists('register_sidebar') ) {
```

Después de eso si refrescamos podremos visualizar en nuestro panel de administrador nuestra nueva área para widgets y plugins en el cual podemos crear y meter lo que así quiéranos.

Ahora falta definirlo para que aparezca en nuestro tema en el front end o lo que es lo que el usuario ve finalmente que es lo que nos interesa para eso vamos a buscar en nuestro archivo donde observemos que se utilizan los div principales de maquetación de la plantilla, para mi caso se encuentra todo definido en el archivo header.php ubicado en \wp-content\themes\Aggregate\header.php en el cual podremos hacer escribir texto plano en cualquier ubicación donde veamos divs e ir ubicando en cual lugar queremos ubicar nuestro nueva area para widgets, en mi caso como l oqueiro colocar arriba del primer menú entonces la ubicación es despues del body pero antes del top header algo asi luce:

```
<body <?php body_class(); ?>>
Aqui colocare mi codigo para que aparezca el area de widget que me interesa
        <?php do_action('et_header_top'); ?>
        <div id="top-header">
```

Ahora bien para pintar el área de widget que creamos en el archivo sidebars.php vamos a tomar lo que pusimos en name en mi caso queda el archivo header.php de la siguiente manera

```
<!—todo el demás código- - >
<body <?php body_class(); ?>>
<div>
    <?php if (!function_exists('dynamic_sidebar') || !dynamic_sidebar('Publicidad Arriba')) : ?>
ANUNCIATE EN MONITOR UNIVERSITARIO, CONTACTANOS...
<?php endif; ?>
     </div>
        <?php do_action('et_header_top'); ?>
        <div id="top-header">
<!—todo el demás código- - >
```

Creando un plugin

La mayoría de los plugins que creemos irán a la carpeta **wp-content/plugins** de nuestro sistema WordPress. Dentro de esta carpeta debemos crear una nueva, y nombrarla con una clave que referencie a nuestro plugin. Como esta es la primera clase de "Crear Plugin para WordPress" y queremos poner un tutorial sencillo, usaremos: wp-content/plugins/mi_plugin. Dentro de nuestra nueva carpeta, crearemos un primer archivo llamado mi-plugin.php

Abrimos este archivo con un editor de código PHP, y escribimos la cabecera. La cabecera son una serie de líneas que aportan información sobre el plugin a WordPress a la hora de mostrarlo en la pantalla de administración, como el nombre, la descripción, la versión, el autor, etc.

Una cabecera simple y básica podría ser como esta:

```
<?php
/*
Plugin Name: Mi plugin
Plugin URI: http://www.web-sistemas.com/
Description: Crear Plugins para WordPress,
Author: Geovanny Quilarque
Author URI: http://www.web-sistemas.com/
Version: 1.0
License: GPLv2
*/
?>
```

Esta cabecera es obligatoria en todos los plugins que hagamos.

Si hemos seguido correctamente los pasos hasta aquí, ya debería aparecer nuestro plugin en la lista de plugins de nuestro WordPress. Eso sí, activarlo o desactivarlo no hace nada, evidentemente, puesto que no hemos programado ningún tipo de funcionalidad.

Dotando de funcionalidad al plugin

A partir de aquí, lo que quieras que tu plugin haga, es totalmente cosa tuya. Depende de su imaginación. Pero no basta con escribir funciones y llamarlas.

WordPress tiene una serie, o mejor dicho, una GRAN serie de "hooks". Ya hemos hablado de ellos alguna vez, pero volveremos a explicar el concepto.

Un "**Hook**" o "Gancho", en WordPress, es un momento en la carga del sistema al que podemos acoplar nuestras funciones. Es decir, por ejemplo, si necesitamos añadir un menu nuevo al administrador de WordPress, programamos la función que corresponde, y después, la asociamos al hook "admin_menu", para que cuando se esté cargando el menú de administración, nuestra

función se ejecute. Muy práctico, ¿Verdad?

Los hooks se llaman durante “acciones” o durante “filtros”, con add_action o con add_filter. ¿Qué diferencia hay?

Acciones: Las acciones son aquellos ganchos que el sistema lanzará en puntos específicos durante la ejecución, o cuando eventos específicos ocurran. Tu plugin puede especificar que sus funciones sean lanzadas en estos puntos, usando la API de acciones.
Filtros: Los filtros son ganchos que WordPress lanza para modificar textos de varios tipos antes de añadirlos a la base de datos, o antes de mostrarlos en pantalla. Tu plugin puede especificar que sus funciones sean ejecutadas para modificar ciertos textos en esos puntos, usando la API de filtros.
En ocasiones, usar una acción o un filtro nos conducirá al mismo resultado, independientemente de lo que usemos. Por ejemplo, si quieres que tu plugin cambie un texto de un post, puedes añadir una acción al hook ‘publish_post’ (que se encargará de modificarlo cuando se guarde en la base de datos), o bien, usar un filtro en el hook ‘the_content’ (y el contenido será modificado al mostrarse en pantalla). Puedes leer más información sobre esto en el Codex.

Si no asociamos una función a un hook, esta no se ejecutará nunca, salvo que durante la ejecución normal del sistema (en un theme, o en otra parte de nuestro plugin) la llamemos directamente.

A la hora de crear funciones, debemos tener mucho cuidado. Si dos plugins implementan la misma función, la consecuencia es un error grave en el sistema que impide que este cargue. Asi pues, se recomienda encarecidamente:

Diferenciar nuestras funciones y variables globales correctamente, inventando algún tipo de prefijo o palabra que haga nuestro plugin único, o lo más único posible. No podemos saber, evidentemente, si otro programador en otra esquina del mundo va a implementar exactamente el mismo nombre de función, con el mismo distintivo, y se va a dar la casualidad en un tercer rincón del planeta de que un usuario instale los dos plugins en su WordPress, de acuerdo. Pero podemos intentarlo. Por ejemplo, en este caso, vamos a llamar a nuestras funciones con el prefijo “mi_plugin”. Bien, es un mal ejemplo, ¿Cuántos plugins empezarán con ‘mi_plugin’? Muchísimos, fijo. Quizá podríamos darle una vuelta, que sea ‘-mip_’, ‘__miplugin_’, ‘codigonexo_mi_plugin’, o lo que se nos ocurra. Recuerda que luego tienes que usarlo, y que cuanto más largo, mas difícil de recordar y más tedioso de reescribir será.

Comprueba que la función no exista antes de definirla. Este paso es mucho más inteligente que el anterior (pero no lo sustituye, que conste). Si la función ya existe, directamente no se define la nuestra. Esto tampoco soluciona el problema que comentabamos antes, pero en este caso, en lugar de provocar un error en el sistema, simplemente no funcionará nuestro plugin. Puedes hacerlo así:

```
if(!function_exists('mi_plugin_function')){
   function mi_plugin_function(){
      // Contenido de la función
   }
}
```

Y luego, a la hora de hacer uso de dicha función en algún punto del plugin, comprobar si dicha función existe, pero a la inversa:

```
if(function_exists('mi_plugin_function')){
   mi_plugin_function();
}
```

Por otro lado, también debemos saber que WordPress cuenta con funciones para (casi) todo en el sistema. ¿Quieres añadir un menu? add_menu_page. ¿Quieres enviar un email? wp_mail. ¿Quieres obtener el título de un post? get_the_title. ¿Te has inventado un nuevo shortcode? add_shortcode. Así que cuando se nos ocurra algo para hacer en el sistema, recuerda estos consejos antes de empezar a programar:

Probablemente, **a alguien se le ha ocurrido antes que a ti**. Y esto no es malo, al contrario: alguien se ha partido la cabeza para conseguirlo, y por lo general, **la solución suele estar en Internet** más o menos escondida.

Si Google es el mejor amigo de un desarrollador, el **Codex de WordPress es el segundo mejor amigo de un desarrollador de WordPress** (el primero sigue siendo Google). No solo por ser la **documentación oficial de WordPress** para desarrolladores, si no porque además **sus foros son ricos en preguntas raras de narices**, que en muchos casos o bien nos aclararán la duda, o bien nos pondrán en la pista. Imprescindible saber inglés, aunque eso es algo que se aplica a toda nuestra profesión, no debería ser un grave problema…

Otro gran sitio donde mirar es **Stack Overflow**, aunque, por lo general, cuando buscas en Google, es de los primeros sitios que sale con respuestas y suele ser más efectivo.

No tengas miedo a **preguntar**.

Si ninguno de estos consejos realmente ha dado solución a tu problema o te lo ha puesto un poco más fácil, recuerda que nada es imposible. **Intenta simplificar el problema**, dividirlo en partes, y busca soluciones a cada parte. No te atasques, no te centres en un asunto al que no le ves solución salvo que no tengas alternativa. Si puedes pasar a otra tarea más sencilla donde despejarte, cuando vuelvas quizá lo veas todo mas claro. Y sobre todo, **nunca te desanimes**.

Ahora vamos a desarrollar un pequeño ejemplo consistente en un submenú dentro del menú "Ajustes" y una página donde poder guardar opciones directamente en la base de datos de WordPress.

El menú

Como ya hemos dicho, WordPress tiene funciones para todo lo "típico" que los plugins hacen. Añadir un submenú a la sección Ajustes entra dentro de lo normal, así que WordPress lo contempla, y podemos hacerlo tal que así:

```
function miplugin_menu_opciones(){

    add_options_page('Título de la pagina','Título del menu','read','miplugin-ops','miplugin_pagina_de_opciones');

}
```

Aunque aquí utilizamos la función "add_options_page", en realidad esta llama a la función genérica de creación de submenús, "add_submenu_page", solo que no tienes que preocuparte por especificar de dónde cuelga este submenú. Los cinco parámetros son: el título de la página (el que aparece arriba del navegador y en la pestaña), el título que aparecerá en el submenú, el permiso mínimo que debe tener el usuario para ver este submenú ('read' es de los más bajos, estaría abierto a todos los usuarios con acceso a wp-admin), un slug único para este menú, y la llamada a la función que se ejecutará cuando pulsemos en este menú.

¡**Ojo**! Si en tu plugin quieres crear un tipo personalizado, **no es necesario que crees un menú específico para él** ya que se creará automáticamente el menú, el submenú de listado de entradas y el submenú de crear nueva entrada. Si quieres añadir más cosas (opciones, otras páginas), sí debes crear submenús (salvo para taxonomías, estas también crean sus propios menús donde corresponde).

¿No funciona? Evidentemente, ¿Qué hemos dicho acerca de los hooks? Añadid esta línea justo debajo de la función.

```
add_action('admin_menu','miplugin_menu_opciones');
```

add_action añade, en el momento en que ocurre 'admin_menu', la ejecución de la función 'miplugin_menu_opciones', es decir, la que pone el submenú en su sitio.

La página de funciones

Al añadir el submenú hemos especificado un parámetro como 'miplugin_pagina_de_opciones'. Esta es la función que se ejecutará al acceder a nuestro submenú. Así que vamos a definirla; en realidad, esta función lo que va a hacer es mostrar un formulario HTML…

```
function miplugin_pagina_de_opciones(){
   ?>
   <div class='wrap'><h2>Opciones de mi sitio</h2></div>

   <form method='post'>
      <input type='hidden' name='action' value='salvaropciones'>
      <table>
         <tr>
            <td>
               <label for='telefono'>Telefono</label>
            </td>
            <td>
               <input type='text' name='telefono' id='telefono' >
            </td>
         </tr>
         <tr>
            <td>
               <label for='direccion'>Dirección</label>
            </td>
            <td>
               <input type='text' name='direccion' id='direccion' >
            </td>
         </tr>
         <tr>
            <td>
               <label for='email'>Email</label>
            </td>
            <td>
               <input type='text' name='email' id='email' >
            </td>
```

```
            </tr>
            <tr>
                <td colspan='2'>
                    <input type='submit' value='Enviar'>
                </td>
            </tr>
        </table>
    </form>
    <?php
}
```

Formulario HTML que, dicho sea de paso, carece de funcionalidad. Si pulsamos en Enviar no pasa nada. Retoquemoslo un poco.

```
function miplugin_pagina_de_opciones(){

    echo("<div class='wrap'><h2>Opciones de mi sitio</h2></div>");

    if(isset($_POST['action']) && $_POST['action'] == "salvaropciones"){
        update_option('miplugin_telefono',$_POST['telefono']);
        update_option('miplugin_direccion',$_POST['direccion']);
        update_option('miplugin_email',$_POST['email']);
        echo("<div class='updated message' style='padding: 10px'>Opciones guardadas.</div>");
    }

    ?>

    <form method='post'>
        <input type='hidden' name='action' value='salvaropciones'>
        <table>
            <tr>
                <td>
                    <label for='telefono'>Telefono</label>
                </td>
                <td>
                    <input type='text' name='telefono' id='telefono' value='<?=get_option('miplugin_telefono')?>'>
                </td>
            </tr>
            <tr>
                <td>
                    <label for='direccion'>Dirección</label>
                </td>
                <td>
                    <input type='text' name='direccion' id='direccion' value='<?=get_option('miplugin_direccion')?>'>
                </td>
            </tr>
            <tr>
                <td>
                    <label for='email'>Email</label>
                </td>
                <td>
                    <input type='text' name='email' id='email' value='<?=get_option('miplugin_email')?>'>
                </td>
            </tr>
```

```
        <tr>
          <td colspan='2'>
            <input type='submit' value='Enviar'>
          </td>
        </tr>
      </table>
   </form>

   <?php
}
```

Ahora sí. Esta es una forma sencilla de hacerlo sin calentarnos mucho la cabeza. Detectará por $_POST si entran nuevos datos. Si es asi, actualizamos las opciones con update_option. Además, los campos se rellenan con get_option si la opción existe y tiene un valor no nulo. Y en cualquier parte de nuestra web podemos recuperar estos valores con la misma función, get_option, pues hemos establecido opciones globales al sistema.

(*Nota: los validadores y las comprobaciones de errores son cosa suya. La función update_option devuelve true o false dependiendo de si los datos se han salvado o no*).