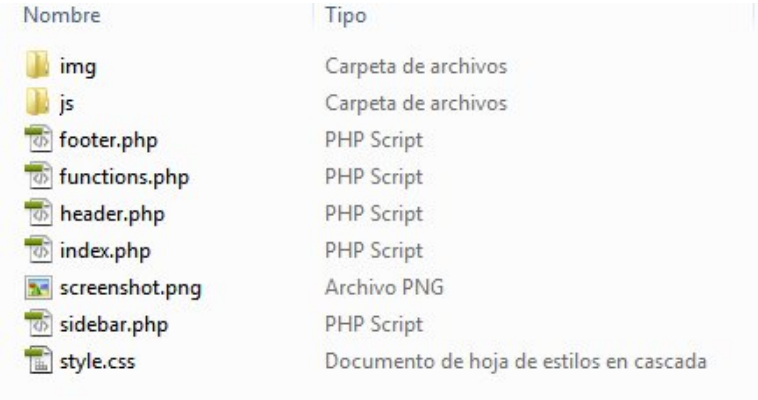
# 1. Estructura de archivos base

Creamos una carpeta con el nombre de la plantilla y dentro creamos la siguiente estructura de archivos:




La filosofía de WP es que el archivo footer.php tenga el código del fin del documento, el functions.php las funciones propias y del WP del tema, el header.php el código del principio del documento y el archivo sidebar.php los contenidos de la barra lateral. La información de estilos se introducirá en el archivo style.css, las imágenes las guardaremos en la carpeta img (el nombre nos lo podemos inventar) y los archivos de javascript los almacenaremos en la carpeta js (el nombre nos lo podemos inventar).

## 1.1. El archivo header.php

Dentro del código HTML introduciremos varios códigos PHP para llamar a las funcionalidades del WP:

1.1.1. Idioma del documento: para definir el idioma del documento como atributo de la etiqueta <html>, llamaremos a la función language_attributes():

```
<html <?php language_attributes(); ?>>
```

1.1.2. Codificación de caracteres: para saber el tipo del documento y el juego de caracteres utilizaremos la función bloginfo( 'html_type' ) para saber los diferentes tipos de información consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/bloginfo:

```
<meta http-equiv="Content-Type" content="<?php bloginfo('html_type');
?>; charset=<?php bloginfo('charset'); ?>" />
```

1.1.3. Título de la página: mediante funciones que detectan el tipo de contenido (consulta el enlace http://codex.wordpress.org/Function_Reference) podemos especificar el título del documento:

```
<title>

<?php if ( is_home() ) { bloginfo('name'); } ?>

<?php if ( is_author() ) { echo 'Archivo por autor'; } ?>

<?php if ( is_single() ) { wp_title(''); } ?>

<?php if ( is_page() ) { wp_title(''); } ?>

<?php if ( is_category() ) { single_cat_title(); } ?>

<?php if ( is_month() ) { the_time('F'); } ?>
```

```
<?php if ( is_search() ) { echo 'Resultados '; } ?>

<?php if ( is_tag() ) { single_tag_title('', true); } ?>

</title>
```

1.1.4. Plugins y otra información de WP: para cargar los scripts de otros plugins o del propio WP utilizamos la función wp_head():

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

1.1.5. La hoja de estilos del tema: para cargar la hoja de estilos CSS del propio tema sin estar pendientes de dónde se ha instalado el WP, utilizamos la función bloginfo('stylesheet_url'):

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

1.1.6. Otros recursos del tema: para cargar otros archivos necesarios para el tema, por ejemplo javascript, sin estar pendientes de dónde se ha instalado el WP, utilizamos la función get_template_directory_uri():

```
<script src="<?php echo get_template_directory_uri();
?>/js/funciones.js" type="text/javascript"></script>
```

1.1.7. Mostrar el título del blog: para mostrar el título del blog introducido en el apartado de Ajustes generales del escritorio del WP utilizamos la función bloginfo( 'name' ), y para añadirle el enlace a la página principal utilizamos la función bloginfo( 'url' ):

```
<a href="<?php bloginfo( 'url' ); ?>"><?php bloginfo( 'name' ); ?></a>
```

1.1.8. Mostrar la descripción corta: para mostrar la descripción corta del blog introducido en el apartado de Ajustes generales del escritorio del WP utilizamos la función bloginfo( 'description' ):

```
<?php bloginfo( 'description' ); ?>
```

1.1.9. Mostrar un formulario de búsqueda: podemos añadir un pequeño formulario de búsqueda, aparte del widget de *Buscar*. Solo hay que configurar la etiqueta <form> con los atributos method="get" action="<?php echo home_url( '/' ); ?>" y el campo de entrada de texto <input> con el atributo name="s":

```
<form method="get" action="<?php echo home_url( '/' ); ?>">

<input type="text" name="s" />

</form>
```

1.1.10. Cargar el menú: para cargar el menú utilizamos la función wp_nav_menu(), pero para que esta función se active hay que añadir la función register_nav_menu() en el archivo function.php. En el archivo funtion.php:

```
<?php register_nav_menu('Principal','Principal'); ?>
```

Y en el archivo header.php:

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

## 1.2. El archivo footer.php

Dentro del código HTML introduciremos varios códigos PHP para llamar a las funcionalidades del WP:

1.2.1. Plugins y otra información de WP: para cargar los scripts de otros plugins o del propio WP utilizamos la función wp_footer():

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

## 1.3. El archivo style.css

En este archivo no solo cargamos la información de estilos CSS de nuestro tema, sino que además contiene la descripción del tema en sí, con lo que al principio del archivo introduciremos las siguientes líneas que describen nuestro tema (el texto en cursiva es personalizable para cada tema):

```
/*
Theme Name: Nombre del tema
Theme URI: http://www.nuestraurl.es
Description: descripción de nuestro tema
Author: Nuestro Nombre
Author URI: http://www. nuestraurl.es
Version: 1.0
Tags: etiquetas, que, definen, nuestro, tema
*/
```

## 1.4. La imagen screenshot.png

Esta es la imagen de muestra de cómo queda nuestro tema, que se muestra en el apartado de Temas del escritorio del WP. Las medidas habituales son 300px de ancho por 225px de alto.

## 1.5. El archivo index.php

Dentro del código HTML introduciremos varios códigos PHP para llamar a las funcionalidades del WP:

1.5.1. Cargar el archivo header.php: el primer contenido del documento index.php debe ser la llamada a la función get_header() para cargar el contenido del documento header.php:

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

1.5.2. Cargar el archivo sidebar.php: para cargar el contenido del documento sidebar.php utilizamos la función get_sidebar(). Según la definición de los estilos css, la llamada a la función debe estar justo después de la llamada a la función get_header() o después de cargar los contenidos. La ventaja de hacer la llamada aparte y no dentro del documento header.php, es que así podemos cargar la barra lateral o no en función del contenido.

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

1.5.3. Cargar otros archivos laterales: si en nuestro tema vamos a tener varios archivos con contenidos laterales, podemos crear tantos archivos *sidebar* como queramos, pero tendrán que tener como nombre la estructura sidebar-*nuevonombre*.php, donde *nuevonombre* será el nombre personalizado que queramos. Para cargar los nuevos *sidebar* pasaremos el *nuevonombre* como parámetro en la función get_sidebar():

```
<?php get_sidebar( 'nuevonombre' ); ?>
```

1.5.4. Listar las entradas: para llamar a todas las entradas publicadas en el escritorio del WP utilizamos la función have_posts(), y para consultar el contenido de la entrada individual utilizamos la función the_post(). Una vez tenemos la entrada individual, con las siguientes funciones llamamos a los diferentes contenidos de la entrada (http://codex.wordpress.org/Template_Tags):

- Titulo de la entrada: the_title()
- URL al contenido de la entrada: the_permalink()
- El contenido de la entrada: the_content( 'Leer más...' ) entre las comillas podemos establecer el texto que hará de enlace si se ha insertado una etiqueta *More*
- El extracto de la entrada: the_excerpt()
- Fecha de creación de la entrada: the_time( get_option( 'date_format' ) ) para ver el formato de fecha y hora consulte el siguiente enlace: http://codex.wordpress.org/Formatting_Date_and_Time
- Hora de creación de la entrada: the_time( get_option( 'time_format' ) ) para ver el formato de fecha y hora consulta el siguiente enlace http://codex.wordpress.org/Formatting_Date_and_Time
- Mirar si la entrada tiene una Imagen Destacada: has_post_thumbnail()
- Mostrar la Imagen Destacada: the_post_thumbnail( 'medium' ) en el documento index.php (para ver las medidas disponibles consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/the_post_thumbnail#Thumbnail_Sizes ) y add_theme_support( 'post-thumbnails' ) en el documento functions.php

```
<?php add_theme_support( 'post-thumbnails' ); ?>
```

- Mostrar el autor: the_author()
- Mostrar la(s) categoría(s) de la entrada: the_category( '*separador*' ) donde *separador* es el carácter que separa los nombres de las categorías. Para ver más opciones consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/the_category
- Mostrar la(s) etiqueta(s) de la entrada: the_tags( '*inicio*', '*separador*', '*fin*' ) donde *inicio* es el texto que aparece antes del listado de las etiquetas, *separador* es el carácter que separa los nombres de las etiquetas y *fin* es el texto que aparece al final del listado de etiquetas. Para ver más opciones consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/the_tags
- Mostrar el enlace de edición: si hay un usuario registrado con permisos para editar las entradas, la función edit_post_link() muestra un enlace para editar la entrada.

```
<?php

if ( have_posts() ) { // condición si hay entradas

  while ( have_posts() ) { // consulto cada una de las entradas

        the_post(); // consulto una entrada particular y

                    //muestro su información

        ?>

        <p><a href="<?php the_permalink(); ?>"><?php the_title();
?></a></p>
```

```
        <?php if ( has_post_thumbnail() ) { ?>

              <p><?php the_post_thumbnail( 'medium' ); ?></p>

        <?php } ?>

        <p><?php the_content(); ?></p>

        <p>Escrito el <?php the_time('F j Y') ?>, por <?php
the_author() ?> en <?php the_category(' '); ?>. <?php
the_tags('Etiquetas: ', ', ', '.'); ?> <?php edit_post_link(); ?></p>

  <?php } // fin de la consulta de las entradas

} else { // si no hay entradas ?>

  <p>No hay entradas para mostrar</p>

<?php } // fin de la condición ?>
```

1.5.5. Cargar la navegación de páginas: si el número de entradas supera el "Número máximo de entradas a mostrar en el sitio" configurado en los ajustes de lectura del escritorio de WP, las funciones previous_posts_link() y next_posts_link() muestran los enlaces para ver las anteriores o las siguientes páginas de entradas; podemos pasar como parámetro el texto que queremos que haga de enlace:

```
<?php previous_posts_link('&laquo; Entradas recientes - '); ?><?php
next_posts_link('Entradas antiguas &raquo;'); ?>
```

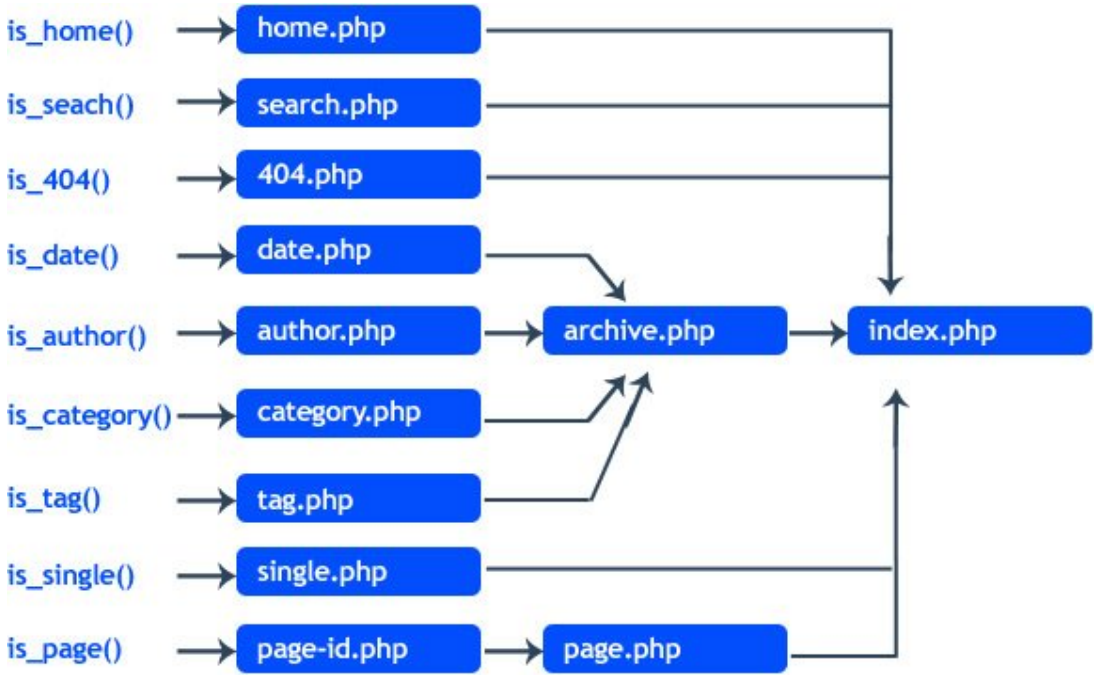
1.5.6. Cargar el archivo footer.php: el último contenido del documento index.php deber ser la llamada a la función get_footer() para cargar el contenido del documento footer.php:

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

## 2. El resto de archivos

Para mostrar todas las entradas, una entrada particular, una lista de entradas de una categoría, una lista de entradas de una etiqueta, una lista de entradas por autor, una página o el resultado de una búsqueda, WP utiliza un documento con un nombre concreto, y si no existe busca por defecto el archivo index.php.

El nombre de los diferentes documentos y su función tiene la siguiente jerarquía:




## 2.1. Categoría individual

El archivo category.php es el que se encarga de mostrar las entradas de una categoría específica (cuando se enlaza desde un menú o desde el widget de *Categorías*). Para mostrar las entradas se utiliza el mismo bucle que en el archivo index.php, pero además podemos mostrar el título y la descripción de la categoría con las funciones single_cat_title() y category_description() respectivamente:

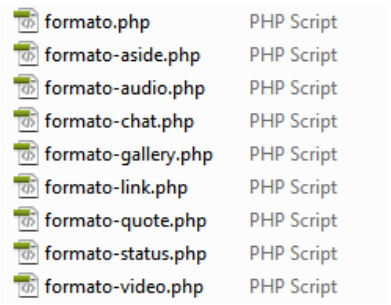
```
<?php single_cat_title(); ?>

<?php echo category_description(); ?>
```

Si queremos mostrar entradas con diferentes formatos en el archivo category.php o en el propio index.php, WP tiene pregenerados 9 formatos más el estándar. Para poder utilizarlos se añade la función add_theme_support() en el archivo function.php con los formatos que queramos utilizar como parámetro:

```
<?php add_theme_support( 'post-formats', array( 'aside', 'gallery', 'link',
'image', 'quote', 'status', 'video', 'audio', 'chat' ) ); ?>
```

De esta forma se muestran en el apartado de Formato del escritorio del WP:

Para cada formato diferente que definamos para las entradas, hay que crear un archivo diferente. Cada archivo tiene como nombre una base común y el nombre del formato separados por un guión medio [-]:




El archivo que solo tiene en nombre común será el formato estándar. Dentro de cada archivo ponemos el diseño y el contenido concreto para una entrada, con las funciones que se ponen después de la función the_post() vistas en el punto 1.5.4.

Finalmente, en el archivo index.php, después de la llamada a la función the_post(), llamamos a la función get_template_part() especificando el nombre de base común de los archivos con los formatos creados:

```
<?php get_template_part( 'formato', get_post_format() ); ?>
```

## 2.2. Etiqueta individual

El archivo tag.php es el que se encarga de mostrar las entradas de una etiqueta específica (cuando se enlaza desde un menú o desde el widget de *Nube de etiquetas*). Para mostrar las entradas se utiliza el mismo bucle que en el archivo index.php o category.php, pero además podemos mostrar el título y la descripción de la etiqueta con las funciones single_cat_title() y category_description() respectivamente (igual que en category.php):

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

<?php echo category_description(); ?>
```

Si estamos utilizando formatos de entrada, también utilizamos la función get_template_part() para mostrar en el bucle los diferentes formatos.

## 2.3.Entrada individual

El archivo single.php es el que se encarga de mostrar el contenido de una entrada individual. La estructura del código puede ser la misma que la del index.php o del category.php, es decir, también podemos utilizar archivos para los diferentes formatos de entrada.

Para añadir los enlaces para las entradas anteriores y siguientes, utilizamos las funciones previous_post_link() y next_post_link() respectivamente:

```
<?php

// NAVEGACION ENTRE ENTRADAS

if ( is_single() ) {

   ?>
```

```
   <section class="navegacion"><?php previous_post_link('&laquo;
%link','Anterior'); ?> - <?php next_post_link('%link &raquo;','Siguiente');
?></section>

<?php } ?>
```

Para ver todas las opciones consulta los siguientes enlaces http://codex.wordpress.org/Function_Reference/previous_post_link y http://codex.wordpress.org/Function_Reference/next_post_link

Además, después de mostrar el contenido podemos añadir la función comments_template() para mostrar todo el apartado de comentarios:

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

En el punto 6 se explica cómo cambiar la apariencia los comentarios.

### 2.4.Página individual

El archivo page.php es el que se encarga de mostrar el contenido de una página. La estructura del código puede ser la misma que la del index.php.

Si queremos mostrar páginas con diferentes diseños, podemos crear diferentes plantillas para las páginas creando archivos que se llamen *nombreplantilla*.php, donde *nombreplantilla* es un nombre que nos podemos inventar. Dentro de este archivo, las primeras líneas deben ser:

```
<?php

/*

 Template Name: Nueva Plantilla

*/

?>
```

Donde *Nueva Plantilla* es el nombre que aparecerá en el apartado de Atributos de página de escritorio de WP:


Y el resto del código sigue la misma estructura que page.php o index.php

## 3. Modificar las consultas

Podemos utilizar la función query_posts() antes de la función have_posts() para modificar qué entradas se cargan a continuación; para ver todas las opciones consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/query_posts:

```
<?php query_posts( 'category_name=blog' ); ?>
```

Una vez finalizado el bucle, hay que utilizar la función wp_reset_query() para restaurar las consultas.

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

# 4. Crear nuevos menús

## 4.1. Trabajar con un solo menú:

Si queremos modificar el menú del tema, primero tendemos que modificar la función register_nav_menu() en el archivo functions.php. Esta función necesita dos parámetros: la localización y la descripción. Ambos son nombres que nos podemos inventar (si utilizamos acentos y caracteres latinos antes deberemos asegurarnos que el archivo functions.php está codificado en UTF-8). Por ejemplo:

```
register_nav_menu('Principal','Menú Principal');
```

Y en el archivo donde queramos que salga el menú (normalmente el header.php) llamamos a la función wp_nav_menu().

## 4.2. Trabajar con varios menús:

Si queremos poner varias zonas de menús en nuestro tema, utilizamos la función register_nav_menus() en el archivo functions.php. Esta función tiene como parámetro un array asociativo con los nombres de las posiciones y sus descripciones:

```
<?php
register_nav_menus( array(
'top' => 'Menú Superior',
'left' => 'Menú Izquierda',
'bottom' => 'Menú Inferior'
) );
?>
```

El nombre de la descripción (*Menú Superior, Menú Izquierda, Menú Inferior*) aparece en el escritorio de WP para poder asignar los menús que se creen en las diferentes posiciones:


El nombre la posición (*top, left y bottom*) se utiliza como parámetro en la función wp_nav_menu() para mostrar el menú seleccionado en los archivos del tema (normalmente en el header.php):

```
<?php wp_nav_menu( array('theme_location' => 'top' ) ); ?>
```

Para ver el resto de parámetros disponibles consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/wp_nav_menu.

Entre los diferentes parámetros para añadir a la función wp_nav_menu() está disponible el parámetro *walker*, que permite modificar la construcción del menú. Por ejemplo, se puede añadir la clase "flecha" a los elementos que tengan submenús: en el archivo de functions.php añadimos una nueva clase Flechas_Menu:

```
class Flechas_Menu extends Walker_Nav_Menu{

   public function display_element($el, &$children, $max_depth, $depth = 0,
$args, &$output){

         $id = $this->db_fields['id'];

         if(isset($children[$el->$id]))

               $el->classes[] = 'flecha';

         parent::display_element($el, $children, $max_depth, $depth, $args,
$output);

   }

}
```

Y en el archivo header.php, en la llamada a la función wp_nav_menu(), añadimos el parámetro *walker*:

```
<?php wp_nav_menu( array('theme_location' => 'top', 'walker' => new
Flechas_Menu() ) ); ?>
```

## 5. Añadir zonas para widgets

Para poder añadir widgets a nuestro tema, primero hay que especificar qué áreas para widgets tendrá el tema en el archivo functions.php con la función register_sidebar(), pasando como parámetros el nombre del área y el código HTML que se genera.

```
register_sidebar(array(
 'name' => 'Lateral',
 'before_widget' => '<div class="widget">',
 'after_widget' => '</div>',
 'before_title' => '<h3>',
 'after_title' => '</h3>',
 ));
```

Para cada región de widgets añadiremos una nueva llamada a la función register_sidebar() cambiando el nombre al parámetro name. Para ver el resto de parámetros disponibles consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/register_sidebar.

Una vez registrado el área para widgets del escritorio, llamamos a los widgets cargados en esa área con la función dynamic_sidebar() en el archivo sidebar.php:

```
<?php dynamic_sidebar( 'Lateral' ); ?>
```

## 6. Modificar la apariencia de los comentarios

Si queremos personalizar la apariencia de los comentarios que se muestran con la función comments_template(), debemos añadir en el archivo style.css los estilos CSS correspondientes:

```
/* Comentarios */
 ol.commentlist {}
 ol.commentlist li {}
 .comment-author {}
 .comment-author img {}
 .fn {}
 .says {}
 .comment-meta {}
 .comment-awaiting-moderation {}
 .comment-meta a {}
 ol.commentlist li p {}
 .reply a {}
```

También podemos modificar la estructura de la información de los comentarios si a la función comments_template() le pasarnos como parámetro el nombre del archivo que tiene la nueva estructura de comentarios precedido por una barra "/" (por defecto llama a /comments.php):

```
<?php comments_template('/comentarios.php'); ?>
```

En el archivo comentarios.php modifico el formulario de envío de comentario para que parezca un formulario de contacto mostrando solo el formulario y no la lista de comentarios:

```
<?php
```

```
$campos =  array(

'author' => '<p class="comment-form-author">' . '<label
for="nombre">Nombre</label> <span class="required">*</span> <input id="nombre"
name="author" type="text" value="" size="30"' . $aria_req . ' /></p>',

'email'  => '<p class="comment-form-email"><label for="email">E-mail</label>
<span class="required">*</span> <input id="email" name="email" type="text"
value="" size="30"' . $aria_req . ' /></p>',

'url'    => '<p class="comment-form-tel"><label for="tel">Teléfono</label> <span
class="required">*</span> <input id="tel" name="url" type="text" value=""
size="30"' . $aria_req . ' /></p>',

);

$param = array(

'title_reply'=>'<h4>Contacte con nosotros</h4>',

'comment_notes_before' => '<small>Los campos necesarios están marcados
*.</small>',

'fields' => $campos,

'comment_notes_after' => '',

'label_submit' => 'Enviar'

);

comment_form($param);

?>
```

Para mostrar el formulario de contacto se utiliza la función comment_form(), que admite parámetros para modificar su apariencia. Para ver el resto de parámetros disponibles consulta el siguiente enlace http://codex.wordpress.org/Function_Reference/comment_form

## 7. Configurar la cabecera

Para dejar la opción de configurar la cabecera del tema se utiliza la función add_theme_support( 'custom-header' ) en el archivo functions.php. Nota: esta función está disponible desde la versión 3.4.

```
<?php

global $wp_version;

if ( version_compare( $wp_version, '3.4', '>=' ) )

      add_theme_support( 'custom-header' );

?>
```

Y en el archivo header.php comprobar si se ha cargado una imagen desde el escritorio de WP con la función get_header_image() y recoger la ruta de la misma con la función header_image().

```
<?php

$imagen_cabecera = get_header_image();

if ( ! empty( $imagen_cabecera ) ) {?>
```

```
      <p><a href="<?php bloginfo('url'); ?>/"><img src="<?php header_image(); ?>"  alt="<?php bloginfo( 'name' ); ?>" title="<?php bloginfo( 'name' ); ?>" /></a></p>

<?php } ?>
```

Para mostrar el título del blog del color configurado en el escritorio del WP, añadimos la clase “titulo” al enlace en el archivo header.php:

```
<a href="<?php bloginfo('url'); ?>/" class="titulo"><?php bloginfo( 'name' ); ?></a>
```

En el mismo archivo header.php, debajo de la llamada a la hoja de estilos del tema (ver punto 1.1.5), añadimos una modificación a la hoja de estilos utilizando la función get_header_textcolor() para consultar el color y la función header_textcolor() para mostrarlo:

```
<?php if ( is_string( get_header_textcolor() ) and get_header_textcolor() != 'blank' ) { ?>

      <style type="text/css">

            a.titulo {color:#<?php header_textcolor(); ?>;}

      </style>

<?php } ?>
```