Child Theme Quick Guide

By: Lance Howell

When trying to make changes to a website, a staggering number of people opt to edit their theme directly. They are changing or adding files in their current theme's folder. The biggest disadvantage is that any modifications made to the theme in this way will be lost once the developer updates the theme. Consequently, users either will not be able to keep their theme up to date.

A much better idea is to use a child theme. This allows you to make any number of changes to a website without touching any of the original theme files. A much better idea is a child theme. This allows you to make any number of changes to a website without touching any of the original theme files.

What are Child Themes and Why Use Them?

A theme only becomes a parent theme when someone builds a child theme for it. Every theme that includes all the files required in order to be considered complete can be a parent theme.

The big difference is that a child theme depends completely on its parent in order to work. That is because a child theme is not a standalone entity, but instead modifies or adds to the files of an existing theme. It changes only those parts that you want to be different.

Advantages of Child Themes

Instead of having to create a complete theme from scratch, you can build on something that already exists, thus speeding up development time.

You can take advantage of the functionality of sophisticated freameworks and parent themes, whie customizing the design to your needs.

You can upgrade the parent theme without losing your customizations.

If you are not satisfied with your customizations, just disable the child theme and everything will be as it was before.

A child theme can contain image folders, JavaScript, CSS, template files and many other things. A child theme only needs three things: a folder, a style sheet and a functions.php file. That is it. And the two files can even pretty much be empty.

When to Use a Child Theme

So, should you always build a child theme whenever you want to make any changes to a WordPress website? If you plan to make only minor modifications, such as color changes or a different font, then a custom CSS plugin might be all you need.

Set Up a Basic Child Theme

Create a Folder In wp-content/themes

A child theme needs three things: its own folder, a style sheet and a functions.php file. Like any theme, child themes are located in wp-content/themes in your WordPress installation. So navigate there now and create a new folder for your child theme.

A best practice is to give your theme's folder the same name as the parent theme and append it with –child. Because we are using the Twenty Seventeen theme, we will call our folder twentyseventeen-child.

Create a Style Sheet

Now that we have our folder, we will need a style sheet. In case you are not aware, a style sheet contains the code that determines the design of a website. Theme can have multiple style sheets, but we will be content with one for the moment.

Making a style sheet is easy: Simply create a new text file and call it style.css. We have to paste the following code, "style sheet header," right at the beginning of the file

/*

Theme Name: Twenty Seventeen Child

Theme URI: http://example.com/twenty-seventeen-child/

description: >-

Twenty Seventeen Child Theme

Author: John Doe

Author URI: http://example.com

Template: twentyseventeen

Version: 1.0.0

License: GNU General Public License v2 or later

License URI: http://www.gnu.org/licenses/gpl-2.0.html

Tags: light, dark, two-columns, right-sidebar, responsive-layout, accessibility-ready

Text Domain: twenty-seventeen-child

*/

Meaning of the Code

- **Theme Name:** This is the name that will show up for your theme in the WordPress back end.
- **Theme URI:** This points to the website or demonstration page of the theme at hand.
- **Description:** This description of your theme will show up in the theme menu when you click on "Theme Details."
- **Author:** This is the author's name—that's you.
- **Author URI:** You can put your website's address here if you want.
- **Template:** This part is crucial. Here goes the name of the parent theme, menaing its folder name. Be aware that it is case-sensitive, and if you don't put in the right information, you will receive an error message.
- **Version:** This displays the version of your child theme.
- **License:** This is the license of your child theme. WordPress themes in the directory are usually released under a GPL license; you should stick with the same license as your parent theme.
- **License URI:** This is the address where your theme's license is explained. Again, stick with what your parent theme says.
- **Tags:** The tags help others find your theme in the WordPres directory. Thus, if you include some, make sure they are relevant.
- **Text Domain:** This part is used for internationalization and to make themes translatable. This should fit the "slug" of your theme.

All you really need is the theme name and template. The rest is important only if you plan to publish your theme.

Create Functions.php

The functions.php file allows you to change and add functionality and features to a WordPress website. It may contain both PHP and native WordPress functions. Plus, you are free to create your own functions. In short, functions.php contains code that fundamentally changes how a website looks and behaves.

Creating the file is as easy as creating a style sheet, if not more so. All you need is a text file named functions.php, and then paste in the following code:

```
<?php

//* Code goes here
```

If you don't plan to use PHP to modify your theme, then you can completely do without it. A style sheet and other files might be enough for you.

Inherit Parent Styles

The recommended way to load the parent's style sheet and the reason why we created functions.php earlier—is to use wp_enqueue_style(). This WordPress function safely adds style sheet files to a WordPress theme. The code looks like this:

```
add_action( 'wp_enqueue_scripts', 'enqueue_parent_styles' );
```

```
function enqueue_parent_styles() {
   wp_enqueue_style( 'parent-style', get_template_directory_uri().'/style.css' );
}
```

Be sure to past this at the beginning of your functions.php file, and save it.

Extras

If you want to get all fancy, you could add a theme image. This image will show up in the theme menu in WordPress. All you need to do is create a PNG file, named screenshot.png, and place it in your theme's folder. Make sure to put it in the top-level directory and not in a subdirectory such as images.

The recommended size is 880x660 pixels, although it will be shown as 387x290. The larger dimensions ensure that the image will show up well on high-resolution screens. Other image formats such as JPEG and GIF would also work, but PNG is recommended.

Customizing Your WordPress Child Theme

Implementing Custom Styles

One of the easiest ways to make changes to your theme is via CSS. This allows you to customize colors, dimensions, fonts and other fundamental design elements.If you are proficient in CSS, youcould actually change the entire layout of your website.

For now, all you need to know is that, with style.css in place, you can override any styles in the parent them by adding code to the child theme's style sheet.

```
/*
 Theme Name:   Twenty Seventeen Child Theme
 description: >-
  A child theme of the Twenty Seventeen default WordPress theme
 Author:       Nick Schäferhoff
 Template:     twentyseventeen
 Version:      1.0.0
*/
// Custom styles go here
```

Let's say you are not a fan and want to cram a few more words into each line. Use a tool such as Firebug or Google Developer tools to figure out which styles need to be modified.

```
.entry-header {
   padding: 0 5%;
```

```
}

.entry-title, .widecolumn h2 {

  margin-bottom: 0.5em;

}

.entry-content, .entry-summary {

  padding: 0 5% 10%;

}
```

Overriding Parent Theme Files

You can not only target individual style declarations via the style sheet, but also override entire components of the parent theme. For every theme file present in the parent directory, WordPress will check whether a corresponding file is present in the child theme and, if so, use that one instead. This means that a header.php file in the child theme will override its equivalent in the parent folder.

So, if you don't like something about a page's layout, just copy the respective file, implement your changes, and upload it to the child theme's folder. The modifications will then appear in the child theme, while the original file will remain untouched.

If we take content.php from the Twenty Seventeen theme's folder and open it with an editor, among others things, we will find the following code:

```
<?php

  // Post thumbnail.

  twentyseventeen_post_thumbnail();

?>


<header class="entry-header">

  <?php

  if ( is_single() ) :

    the_title( '<h1 class="entry-title">', '</h1>' );

  else :

    the_title( sprintf( '<h2 class="entry-title"><a href="%s" rel="bookmark">', esc_url( get_permalink() ) ),
'</a></h2>' );

  endif;

  ?>
```

```
</header><!-- .entry-header -->
```

Let's see what happens when we reverse the order of these two, like this:

```
<header class="entry-header">
  <?php
    if ( is_single() ) :
      the_title( '<h1 class="entry-title">', '</h1>' );
    else :
      the_title( sprintf( '<h2 class="entry-title"><a href="%s" rel="bookmark">', esc_url( get_permalink() ) ), '</a></h2>' );
    endif;
  ?>
</header><!-- .entry-header -->


<?php
  // Post thumbnail.
  twentyseventeen_post_thumbnail();
?>
```

Working with Template Files

We've learned that we can overwrite any file in the parent theme by placing a copy in the child theme's folder and customizing it. Let's say we want to build a full-width page template for our child theme. To create a full-width page in Twenty Seventeen, we need to do 4 things: create a custom page template, a custom header and a footer file, and then add some customized CSS.

For our custom page template, we simply copy page.php from the parent theme, rename it to custom-full-width.php and place it in a folder named page-templates in our child theme.

```
<?php
/*
 * Template Name: Custom Full Width
 * description: >-
  Page template without sidebar
 */
```

```
get_header('custom'); ?>

  <div id="primary" class="content-area">
  <main id="main" class="site-main" role="main">

  <?php
  // Start the loop.
  while ( have_posts() ) : the_post();

  // Include the page content template.
  get_template_part( 'content', 'page' );

  // If comments are open or we have at least one comment, load up the comment template.
  if ( comments_open() || get_comments_number() ) :
    comments_template();
  endif;

  // End the loop.
  endwhile;
  ?>

  </main><!-- .site-main -->
  </div><!-- .content-area -->

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

The only thing we have done here is introduce a header that tells WordPres that this is a page template, and we have changed the get_header and get_footer calls so that they will include two files named header-custom.php and footer-custom.php.

Now it’s time to create our custom header and footer theme files. First, go to the parent theme, copy both hearder.php and footer.php to our child theme’s folder, and rename them to header-custom.php and footer-custom.php, respectively.

Let us start with our custom header.

```
<?php
/**
 * The template for displaying the header
 *
 * Displays all of the head element and everything up until the "site-content" div.
 *
 * @package WordPress
 * @subpackage Twenty_Fifteen
 * @since Twenty Fifteen 1.0
 */
?><!DOCTYPE html>
<html <?php language_attributes(); ?> class="no-js">
<head>
  <meta charset="<?php bloginfo( 'charset' ); ?>">
  <meta name="viewport" content="width=device-width">
  <link rel="profile" href="http://gmpg.org/xfn/11">
  <link rel="pingback" href="<?php bloginfo( 'pingback_url' ); ?>">
  <!--[if lt IE 9]>
  <script src="<?php echo esc_url( get_template_directory_uri() ); ?>/js/html5.js"></script>
  <![endif]-->
  <?php wp_head(); ?>
</head>

<body class="full-width-body" <?php body_class(); ?>>
<div id="page" class="hfeed site">
```

```
  <a class="skip-link screen-reader-text" href="#content"><?php _e( 'Skip to content', 'twentyfifteen' ); ?></a>


  <header id="masthead" class="site-header full-width" role="banner">

  <div class="site-branding full-width">

  <?php

    if ( is_front_page() && is_home() ) : ?>

      <h1 class="site-title"><a href="<?php echo esc_url( home_url( '/' ) ); ?>" rel="home"><?php bloginfo( 'name' ); ?></a></h1>

      <?php else : ?>

      <p class="site-title"><a href="<?php echo esc_url( home_url( '/' ) ); ?>" rel="home"><?php bloginfo( 'name' ); ?></a></p>

      <?php endif;


    $description = get_bloginfo( 'description', 'display' );

    if ( $description || is_customize_preview() ) : ?>

      <p class="site-description"><?php echo $description; ?></p>

    <?php endif;

  ?>

  <button class="secondary-toggle"><?php _e( 'Menu and widgets', 'twentyfifteen' ); ?></button>

  </div><!-- .site-branding -->

  </header><!-- .site-header -->


  <div id="content" class="site-content full-width">
```

We have done a number of things here. We have given the <body> element a custom class, named full-width-body. We have also added a full-width class to site-header, site-branding and site-content, so that we can assign them custom CSS.

We have gotten rid of all sidebar elements (both the sidebar div and the call to get_sidebar), because we do not want them on our full-width page. The only change we have made in footer-custom.php is to add the full-width class to the footer element, like so:

```
<footer id="colophon" class="site-footer full-width" role="contentinfo">
```

All that's left to do is input some code in our style sheet:

```
.full-width-body::before {
  display: none;
}

.site-content.full-width {
  float: none;
  margin: 0 auto;
}

.site-header.full-width {
  background-color: #fff;
  box-shadow: 0 1px 0 rgba(0, 0, 0, 0.15);
  margin: 0;
  padding: 2% 0;
}

.site-branding.full-width {
  margin: 0 auto;
  width: 58.8235%;
}

.site-footer.full-width {
  float: none;
  margin: 0 auto;
}
```

That is our full-width page.

Using Functions.php

The child theme's functions.pp file is loaded in addition to the file of the same name in the parent theme. It is executed right before the parent theme's function.php—unlike style.css, which replaces the original file. Do not copy the full contents of your parent theme's functions.php file to the file in your child theme.

I want to add a widget area to the footer of the website. We first need to register a widget in our functions.php file.

```
<?php


register_sidebar( array(
  'name'         => 'Footer Widget',
  'id'           => 'footer-widget',
  'before_widget' => '<div class="footer-widget">',
  'after_widget'  => '</div>'
) );
```

This will make the newly created widget area show up in our back end. However, for it to be usable on the website, we need to add the following code to footer.php:

```
<?php if ( is_active_sidebar( 'footer-widget' ) ) :
    dynamic_sidebar( 'footer-widget' );
  endif;
?>
```

Once more, we will copy footer.php from the Twenty Seventeen parent theme and paste it in our child theme. We need to add the call to our new footer widget so that it looks like this:

```
<?php
/**
 * The template for displaying the footer
 *
 * Contains the closing of the "site-content" div and all content after.
 *
 * @package WordPress
```

```
 * @subpackage Twenty_Fifteen
 * @since Twenty Fifteen 1.0
 */
?>

  </div><!-- .site-content -->

  <footer id="colophon" class="site-footer" role="contentinfo">
    <div class="site-info">
    <?php if ( is_active_sidebar( 'footer-widget' ) ) :
        dynamic_sidebar( 'footer-widget' );
      endif;
    ?>
      <?php
        /**
         * Fires before the Twenty Fifteen footer text for footer customization.
         *
         * @since Twenty Fifteen 1.0
         */
        do_action( 'twentyfifteen_credits' );
      ?>
      <a href="<?php echo esc_url( __( 'https://wordpress.org/', 'twentyfifteen' ) ); ?>"><?php printf( __( 'Proudly powered by %s', 'twentyfifteen' ), 'WordPress' ); ?></a>
    </div><!-- .site-info -->
  </footer><!-- .site-footer -->

</div><!-- .site -->

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

</body>

</html>