WordPress Child Themes: Refresher

Every so often I want to explore new WordPress themes for my blog without having to start from scratch using Underscores. At the same time I want the flexibility of customizing the theme without loosing the changes every time I update the theme.

That's where child themes come in. They allow developers to customize an existing theme independently of the parent and without loosing the changes when we update the parent theme.

Process

Creating a child theme is a four-step process:

1. Create a folder inside the `wp-content/themes` directory
2. Create a `style.css` stylesheet and add the theme boilerplate
3. Create a `functions`.php file and add the code to enqueue parent and child stylesheets
4. Activate the theme

For this post we'll use Twenty Twenty as the parent theme to take advantage of the new Gutenberg editor (whether it works or not is a subject for another post).

Create child theme folder

Depending on how you do development creating the theme folder can be done either through a GUI (Windows Explorer or Finder) or through a terminal (Linux, Mac or WSL for Windows)

I work on Mac's terminal so, starting on the root of the WordPress installation I run the following command

```
mkdir -p wp-content/themes/twentytwenty-rivendellweb/
```

Change to the directory you just created.

```
cd wp-content/themes/twentytwenty-rivendellweb/
```

Create a stylesheet: `style.css`

The core of the child theme is the `style.css` stylesheet and the comment block that defines the theme.

The example below is what I used for my Twenty Twenty child theme. Individual fields are explained below.

```
/*
 Theme Name:   Twenty Twenty Rivendellweb
 Theme URI:    https://github.com/caraya/2020-child
 Description:  Twenty Twenty Child Theme for The Publishing Project
 Author:       Carlos Araya
 Author URI:   https://publishing-project.rivendellweb.net/
 Template:     twentytwenty
 Version:      1.0.0
 License:      MIT
 License URI:  https://opensource.org/licenses/MIT
 Tags:         light, dark, two-columns, right-sidebar, responsive-layout
 Text Domain:  twentytwentyrivendellweb
*/
```

Theme name (***REQUIRED***). This is the name that will show up for your theme in the WordPress admin screen

Theme URI. This points to the website or demonstration page of the theme at hand. **This or the author's URI must be present in order for the theme to be accepted into the WordPress directory**.

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.

Author URI. Author's website.

Template (***REQUIRED***). This is the name of the parent theme, meaning 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. Using semver is usually a good way to version your theme

License. This is the license of your child theme. WordPress themes in the directory are usually released under a GPL license.

Licensing is something that it's important to me and needs clarification.

Since I don't plan on releasing my themes through the directory, I can use whatever license I choose. I don't think that the GPL is a good license to use due to its viral nature and GPL compatible licenses, like MIT, are fine too, contrary to what Matt says.

However, since WordPress Core is licensed under GPL it may make more sense to have only one license throughout the codebase. It ends up being a matter of preference.

The Free Software Foundation maintains a list of free software licenses that are compatible with the GPL in case that the GPL is too restrictive for your taste.

License URI. This is the address where your theme's license is explained. The URI must match the license you choose too use.

Tags. The tags help others find your theme in the WordPress 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.

Enqueue parent styles

The recommended way of enqueuing the parent theme stylesheet currently is to add a `wp_enqueue_scripts` action and use wp_enqueue_style() in your child theme's `functions.php`.

The following example adds both the parent and the child theme stylesheets.

```
<?php
function my_theme_enqueue_styles() {

    $parent_style = 'pare'parent-style'// This is 'twentytwenty-style' fo

    wp_enqueue_style( $parent_style, get_template_directory_uri() . '/sty
    wp_enqueue_style( 'child-style',
        get_stylesheet_directory_uri() . '/style.css',
        array( $parent_style ),
        wp_get_theme()->get('Version')
    );
}
add_action( 'wp_enqueue_scripts', 'my_theme_enqueue_styles' );
```

In this example, each theme has a single stylesheet. If a theme has more than one stylesheet you're responsible to enqueue the styles in the right order so the theme will continue working.

A solution is to concatenate the stylesheets during the build step or using SASS imports.

Activate the new theme

Now we can activate the theme from the Administrator interface. Go to `Appearance > Themes`, select the theme that we just created and activate it.

If everything worked out OK, there should be no difference between the parent and child themes.

We can now start working with our child theme and change it to our heart's content and everything should work the same.

Once WordPress detects you're working with a child theme it will no longer warn you about changing the theme's CSS but it will still suggest you use the theme customizer.

Adding additional tools and functionalityh

Now that we have a working child theme we can start looking at more advanced ideas for our theme.

We'll look at three ways to enhance a WordPress theme:

- Using third-party scripts and tools
- Customizing with Gutenberg blocks
- Using CSS to modify the theme appearance
- Modifying PHP templates

Third-part scripts

Warning

This example, and any other function that adds scripts and stylesheets to pages of a WordPress installation that convert to AMP, is likely to run afoul of AMP validation.

If you want to use an AMP compliant syntax highlighter, I'd suggest you look at Syntax-highlighting Code Block (with Server-side Rendering) by Weston Router.

We can create multiple `wp_enqueue_scripts` actions and use wp_enqueue_script() and wp_enqueue_style() to add tools in your child theme's `functions.php` in a similar way to how we added the stylesheets for the theme to work.

For example, let's say that we want too enqueue Prism core and styles for our theme rather than use a plugin.

The file names for the scripts and the stylesheets in the example do not correspond to the real names. In an ideal world, I'd create brand new downloads and organize them inside the theme as needed.

```
<?php
function rivendellweb_enqueue_prism() {

    wp_enqueue_style( 'prism_style', get_template_directory_uri() . '/pris
    wp_enqueue_script( 'prism_script',
        get_template_directory_uri() . '/prism/prism-core.js' );
}
add_action( 'wp_enqueue_scripts', 'rivendellweb_enqueue_prism' );
```

The usual caveats of adding scripts and stylesheets apply. If you need to, use Scripts To Footer and Scripts-To-Footer Exclude AMP to move the scripts to the footer excluding AMP-related scripts.

This should improve performance.

Customizing the theme: Gutenberg Blocks

In addition to customizing the theme itself, we can choose to create custom Gutenberg blocks to fit the type of content we want to create and that we're too lazy to figure out how to do without blocks.

- Gutenberg Blocks
- CoBlocks
- Stackable
- Atomic Blocks.
- Advanced Gutenberg

I would only recommend this for people who are just starting to work with WordPress. Both my use cases use HTML that doesn't work with Gutenberg without changing the way I write… and I'm not going to change it.

Overriding the parent theme: CSS

The first means to change the parent theme is to use CSS in the child theme to override the parent's CSS.

One thing that has changed since I last played with child themes is that WordPress is now based on blocks so you can't just override the specific class.

The following code sets the default content width to 960 pixels.

I had a hard time parsing the first item of the rule. It reads as:

Select any element that has the class `entry-content` **that has any children that *do not have* any of** `alignwide`**,** `alignfull`**,** `alignleft`**,** `alignright`**, or** `is-style-wide` **classes**

We also pick the elements with class, `post-meta-wrapper` or `post-meta`

```
.entry-content > *:not(.alignwide):not(.alignfull):not(.alignleft):not(.a
.post-meta-wrapper,
.post-meta {
    max-width: 960px;
}
```

Again, more research is needed, particularly when working with Gutenberg blocks and what additional classes they introduce to the different components of the page.

Overriding the parent theme: PHP

The hardest way to modify the parent theme is to change the PHP templates to create a new structure or create new templates to represent new types of content.

The basic template file gives the new template a name, `Project Template` and tells WordPress what type of content to use.

```
<?php
/*
Template Name: Project Template
Template Post Type: post, page
*/
get_template_part( 'singular' );
'singular'
```

Once we have the basic theme, we can leverage template partials to further customize the template. Inside the partials we can write custom HTML or template tags to customize the behavior of the template.

The example below changes the header of all pages that don't use the project template. If the page uses the project template then the custom header will not be used, we should provide a fallback with the default template so the site will look as it did before we made the changes…

```
    <?php
        if (! is_page_template(array('temp'templates/template-project.php
        ?>
        <header id="site-header" class="header-footer-group" role="banner
        <p class="blog-title"><strong><a href="<?php bloginfo('url''url'?

            ___PHP4___
```

Likewise, we customize the footer template to remove all content from the footer so we can fully customize it with Gutenberg blocks or we can edit it and customize it with PHP, HTML and CSS.

```
<?php
if ( ! is_page_template( array( 'temp'templates/template-canvas.php') {
?>
    <footer id="site-footer" role="contentinfo" class="header-footer-grou
    <!-- Build footer content here -->
    </footer></footer><!-- #site-footer -->
<?php } ?>
    <?php wp_footer(); ?>

    </body>
</html>
```

See Template Files in the WordPress Theme Handbook.

You can also see a fully worked version of the code in these examples on Github at https://github.com/caraya/2020-child/

Conclusion

We've just looked at the basics on creating and customizing a child theme for

WordPress.

Since the release of WordPress 5.0 and the Gutenberg editor the development procoess has changed significantly and this post barely scratches the surface of what you can do.

A poossible future project is to use WordPress to build a site combining the blog, the project list and the layout experiments site.