Understanding and Using WordPress Child Themes

Key Takeaways:

- A *child theme* is a specially registered WordPress theme that *inherits* most of its properties from a declared *parent theme*. A major reason to create a child theme is to be able to update the parent theme without losing desired customizations.
- `style.css` is the only mandatory file in a child theme. The general rule is that child theme files override parent theme files if they exist.
- To properly use child themes, you'll need to know how to include the parent theme's `style.css`; how the WordPress template hierarchy responds to child themes; and under what circumstances you can overwrite existing functions in `functions.php`.

A child theme is a WordPress theme that depends on a "parent" theme—which could be any other theme already in existence, as long as that theme isn't itself a child theme.

A child theme *inherits* everything about its parent—and then it only changes what it needs to change. So you could set up a child theme that's identical to its parent, simply by importing the parent theme's stylesheet (we'll cover that later) and making no other changes.

Why Use a Child Theme?

If you make changes to a theme, and then update the theme, your changes are gone.

Themes often update, and for very good reasons: important new features, bugfixes, and (especially) crucial security patches, as well as other reasons. There's no way to update "just part of" a theme—so if you make changes to a third-party theme (say a commercial theme) and then update that theme to a new version, your changes are gone.

Child themes let you make only the changes you need to make, all while letting the parent theme continue to update under you. This is the major reason why they exist.

As you get used to working with child themes, you'll discover another reason to use them: they're really convenient. It's much easier to write only the changes you need to a clean file (for example, `style.css` or `functions.php`) than it is to dive into an existing theme and start pulling wires. So child themes are not only necessary, they're nice.

How to Make a Child Theme

To make a child theme, you create a new theme. Normal themes (themes that can become parent themes) have two required files: `style.css` and `index.php`. Child themes, however, have only one: `style.css`. This is because they inherit the parent's `index.php` by default.

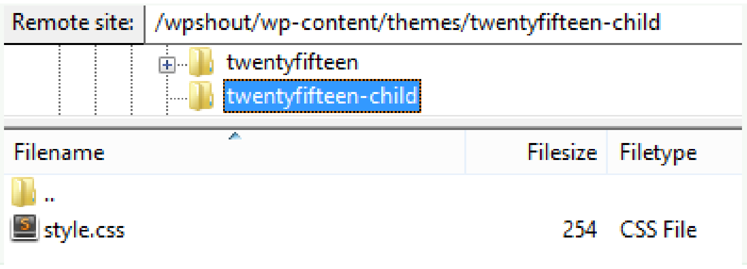
Registering a Child Theme and the "`Template: `" Line

Child themes are registered like normal themes, in a comment block in `style.css`. There's one new field, though: `Template: `. This line is mandatory for every child theme, and corresponds to the name of the *folder* in which the parent theme is located.

Here's an example child theme comment block:

```
/*
Theme Name:     Twenty Fifteen Child
Theme URI:      http://upandrunningwp.com/
Description:    Child theme for the Twenty Fifteen theme
Author:         WPShout
Author URI:     http://wpshout.com/
Template:       twentyfifteen
Version:        1.0.0
*/
```

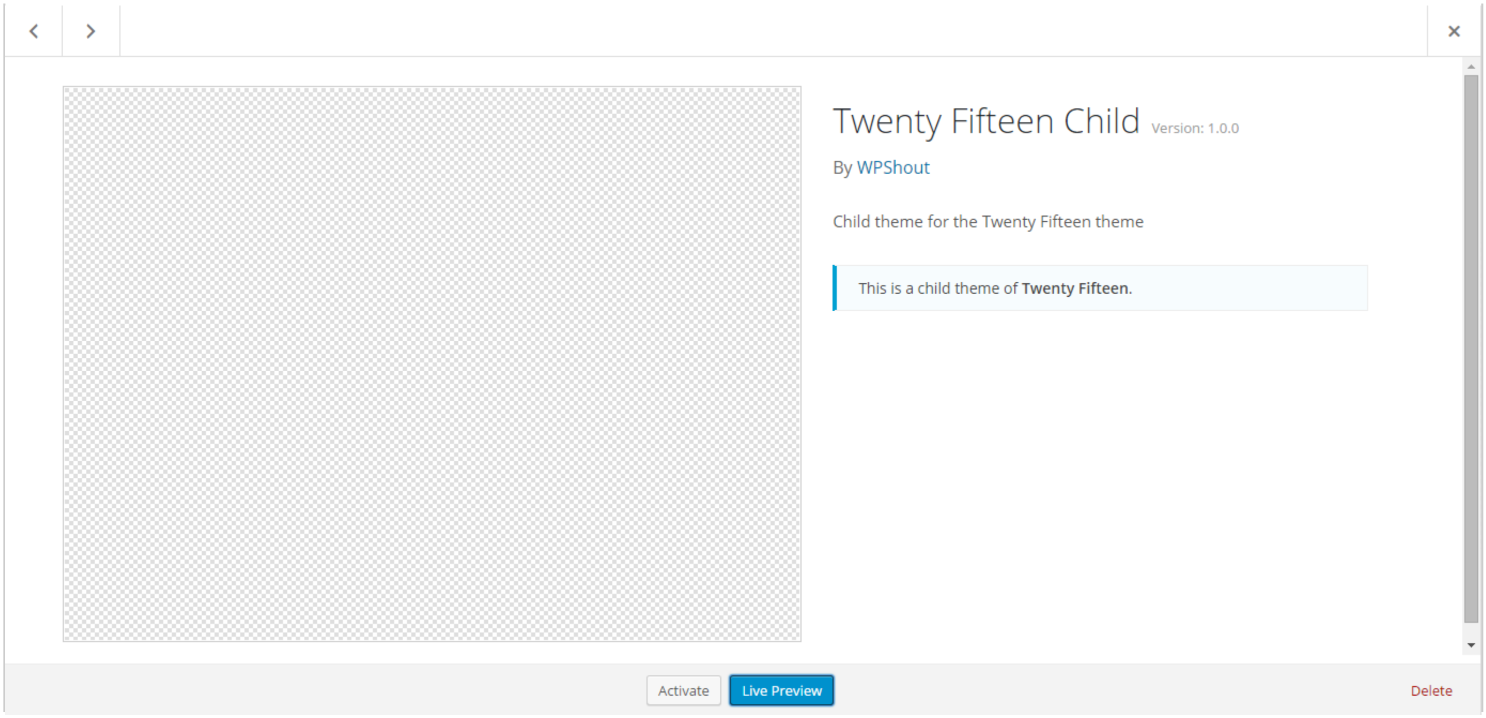
This goes in `style.css` in a new theme folder:



```
Remote site: /wpshout/wp-content/themes/twentyfifteen-child
    twentyfifteen
    twentyfifteen-child

Filename                         Filesize  Filetype
..
style.css                             254  CSS File
```

And gives this result:


Twenty Fifteen Child Version: 1.0.0

By WPShout

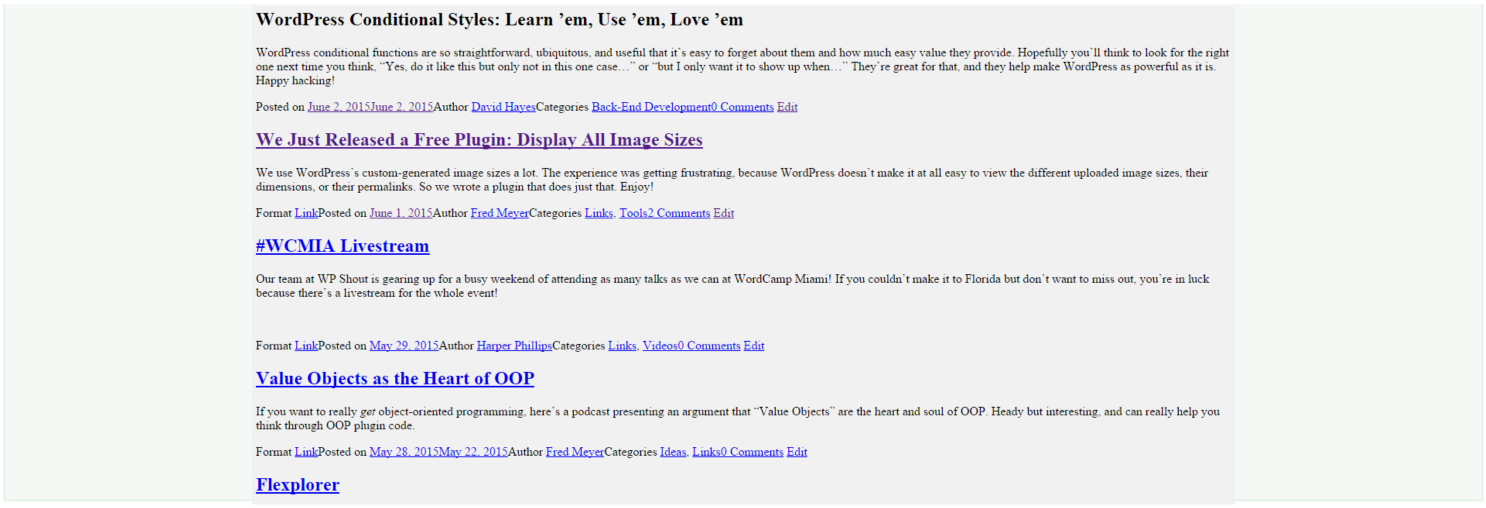
Child theme for the Twenty Fifteen theme

This is a child theme of **Twenty Fifteen**.

Notice, again, that the parent theme's "nice name" is *Twenty Fifteen*, but the name of its *directory* is *twentyfifteen*. That's the one we need to write after `Template:` .

Pulling in the Parent Theme's Stylesheet

If we activate our Twenty Fifteen Child theme right now, our site won't look anything like Twenty Fifteen. On the contrary, it'll look like an absolute disaster:


WordPress Conditional Styles: Learn 'em, Use 'em, Love 'em

WordPress conditional functions are so straightforward, ubiquitous, and useful that it's easy to forget about them and how much easy value they provide. Hopefully you'll think to look for the right one next time you think, "Yes, do it like this but only not in this one case…" or "but I only want it to show up when…" They're great for that, and they help make WordPress as powerful as it is. Happy hacking!

Posted on June 2, 2015June 2, 2015Author David HayesCategories Back-End Development0 Comments Edit

We Just Released a Free Plugin: Display All Image Sizes

We use WordPress's custom-generated image sizes a lot. The experience was getting frustrating, because WordPress doesn't make it at all easy to view the different uploaded image sizes, their dimensions, or their permalinks. So we wrote a plugin that does just that. Enjoy!

Format LinkPosted on June 1, 2015Author Fred MeyerCategories Links, Tools2 Comments Edit

#WCMIA Livestream

Our team at WP Shout is gearing up for a busy weekend of attending as many talks as we can at WordCamp Miami! If you couldn't make it to Florida but don't want to miss out, you're in luck because there's a livestream for the whole event!

Format LinkPosted on May 29, 2015Author Harper PhillipsCategories Links, Videos0 Comments Edit

Value Objects as the Heart of OOP

If you want to really *get* object-oriented programming, here's a podcast presenting an argument that "Value Objects" are the heart and soul of OOP. Heady but interesting, and can really help you think through OOP plugin code.

Format LinkPosted on May 28, 2015May 22, 2015Author Fred MeyerCategories Ideas, Links0 Comments Edit

Flexplorer

Why? Because WordPress is *no longer listening* to the parent theme's `style.css`. The *child's* `style.css` is the primary stylesheet for the site now—and since it has nothing in it, the site looks (and is!) totally unstyled.

There are several good ways to solve this problem; we'll cover two.

@importing the Parent Theme's `style.css` from the Child Theme's `style.css`

This first solution is a single line in the child's `style.css`:

```
@import url("../twentyfifteen/style.css");
```

That's better! This simply uses CSS's `@import` directive to pull *all* the contents of the parent theme's `style.css` file into your child theme's `style.css`. Now your site's looking the way it did before, and you can write your own styles below.

enqueueing the Parent Theme's `style.css` in the Child Theme's `functions.php`

This solution uses a good old `wp_enqueue_style()` call, in the `functions.php` of the child theme:

```
/* Environment: functions.php of a child theme whose parent is twentyfifteen */

function wpshout_enqueue_twentyfifteen_stylesheet( ) {
    wp_enqueue_style(
        'twentyfifteen',
        get_template_directory_uri() . '/style.css'
    );
}
add_action( 'wp_enqueue_scripts', 'wpshout_enqueue_twentyfifteen_stylesheet' );
```

One thing to note here is the use of the function `get_template_directory_uri()`. This function refers to the root directory of the *current* theme if called in a normal theme, and to the root directory of the *parent* theme if called in a child theme. So with this function you don't always know whether you're talking to your own theme or its parent—not too useful in general, but it's what we need here, since we for sure want to be talking to the parent.

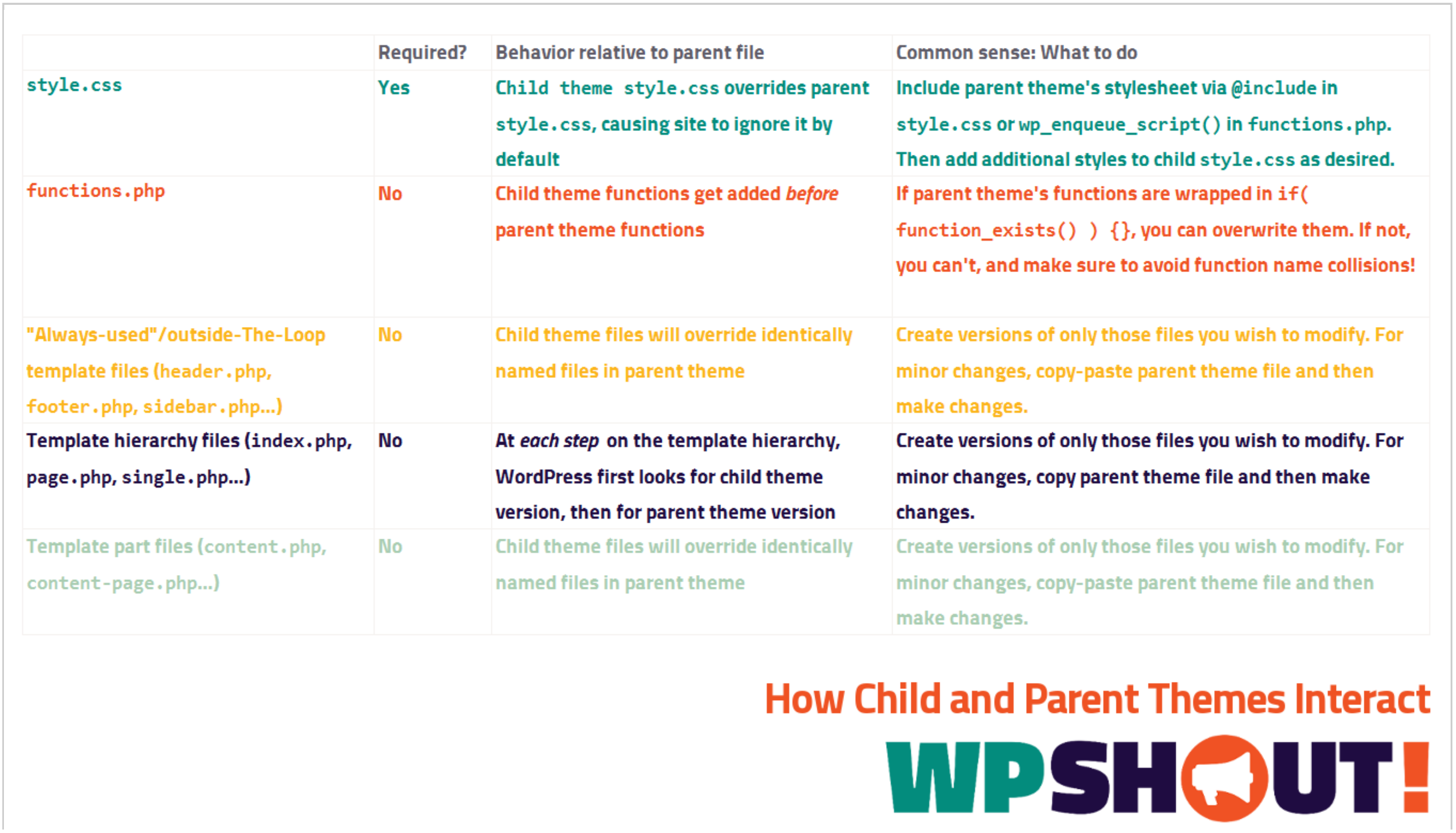
Why would you load the parent's `style.css` in this way? The PHP route turns out to be a little bit quicker, because CSS `@imports` force additional server requests. Remember, it's your *browser*, not the server, that can figure out what `@import` even means. When it does, it has to then turn around and ask the server—a second time—for the stylesheet it's supposed to be `@import`ing.

This speed differential is not a crucial point. Either of these methods is a great way to get your parent themes loaded up.

How Child Theme and Parent Theme Files Interact

We've already seen that the child's `style.css` takes over for the parent's. Let's look more closely at an important question: under what circumstances do the child theme's files substitute for the parent theme's?

Here's a diagram—also in Resources as *How Child and Parent Themes Interact*:



	Required?	Behavior relative to parent file	Common sense: What to do
`style.css`	Yes	Child theme `style.css` overrides parent `style.css`, causing site to ignore it by default	Include parent theme's stylesheet via `@include` in `style.css` or `wp_enqueue_script()` in `functions.php`. Then add additional styles to child `style.css` as desired.
`functions.php`	No	Child theme functions get added *before* parent theme functions	If parent theme's functions are wrapped in `if( function_exists() ) {}`, you can overwrite them. If not, you can't, and make sure to avoid function name collisions!
"Always-used"/outside-The-Loop template files (`header.php`, `footer.php`, `sidebar.php`...)	No	Child theme files will override identically named files in parent theme	Create versions of only those files you wish to modify. For minor changes, copy-paste parent theme file and then make changes.
Template hierarchy files (`index.php`, `page.php`, `single.php`...)	No	At *each step* on the template hierarchy, WordPress first looks for child theme version, then for parent theme version	Create versions of only those files you wish to modify. For minor changes, copy parent theme file and then make changes.
Template part files (`content.php`, `content-page.php`...)	No	Child theme files will override identically named files in parent theme	Create versions of only those files you wish to modify. For minor changes, copy-paste parent theme file and then make changes.

How Child and Parent Themes Interact

WPSHOUT!

In most cases, child theme files simply override identically-named files in the parent.

In most cases, child theme files simply override identically-named files in the parent. The two noteworthy exceptions are:

1. Files in the WordPress template hierarchy
2. `functions.php`

Files in the WordPress Template Hierarchy

When you add a child theme, WordPress still steps through the template hierarchy as it always does, with one difference: at *every step* in the hierarchy, it first checks in the child theme, then in the parent theme.

Traversing the Template Hierarchy with a Child Theme: First Example

Let's say we're about to display our About page, a Page. (Since it'll come up: the Page's `slug` is `about` and its `id` is `2`.)

In this example, the parent theme has two template files: `page.php` and `index.php`. The child theme has one template file: `index.php`.

Here's how WordPress will process:

1. `page-$slug.php`:
    1. Does the child theme have a file named `page-about.php`? (No)
    2. Does the parent theme have a file named `page-about.php`? (No)
2. `page-$id.php`:
    1. Does the child theme have a file named `page-2.php`? (No)
    2. Does the parent theme have a file named `page-2.php`? (No)
3. `page.php`:
    1. Does the child theme have a file named `page.php`? (No)
    2. Does the parent theme have a file named `page.php`? (Yes)

In this case, WordPress will use the parent theme's `page.php` to display the page.

A Second Example

Now let's imagine the child theme does have a `page.php` as well:

1. `page-$slug.php`:
    1. Does the child theme have a file named `page-about.php`? (No)
    2. Does the parent theme have a file named `page-about.php`? (No)
2. `page-$id.php`:
    1. Does the child theme have a file named `page-2.php`? (No)
    2. Does the parent theme have a file named `page-2.php`? (No)
3. `page.php`:
    1. Does the child theme have a file named `page.php`? (Yes)

In this case, the child theme's `page.php` will override the parent's, and WordPress will use the child's `page.php` to display the page.

`functions.php`

A child theme's `functions.php` loads *before* its parent's.

`functions.php` is a very special case: it loads *before* the parent's `functions.php`. Depending on how the parent theme is written, this can play out in two very different ways:

Themes with Pluggable Functions

Careful parent theme authors use this check to let the child's `functions.php` overwrite specific parent theme functions.

PHP has a function titled `function_exists()`. Much like WordPress's conditional tags, it returns a boolean—either `true` or `false`—depending on whether the function it's given to look for exists or not.

Careful parent theme authors wrap all their functions in `if ( ! function_exists() ) {}`. In plain language, this means "if function doesn't exist": the `!` means "not," and so the if-statement is `false` if the function *does* exist, and `true` otherwise. Theme developers use it as follows:

```php
/* Environment: Parent theme's functions.php */

// Only run if wpshout_filter_example() does not already exist
if ( ! function_exists( 'wpshout_filter_example' ) ) {
    function wpshout_filter_example( $title ) {
        return 'Hooked: '.$title;
    }
    add_filter( 'the_title', 'wpshout_filter_example' );
}
```

`if ( ! function_exists( ) )` checks create what are called *pluggable functions*.

This creates what are called *pluggable functions*: functions that the child theme's `functions.php` can overwrite. How does the child theme overwrite them? Simply by registering those functions, since the child's `functions.php` executes first! So the child theme author could write:

```
/* Environment: Child theme's functions.php */

function wpshout_filter_example( $title ) {
    return 'Hooked by Child! ' . $title;
}
add_filter( 'the_title', 'wpshout_filter_example' );
```

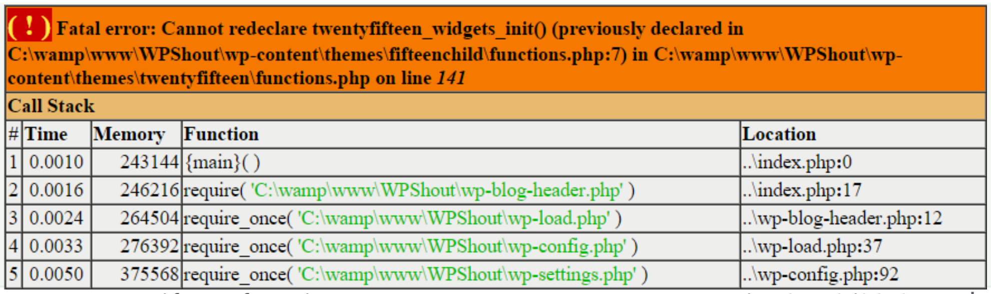
Here, the child's `wpshout_filter_example()` will execute—and the parent's *won't*, since `! function_exists( 'wpshout_filter_example' )` now evaluates to `false`.

Themes Without Pluggable Functions

This situation makes life more difficult for child theme authors. Let's take our same example: the child theme `functions.php` registering a function called `wpshout_filter_example()` that also exists in the parent theme's `functions.php`.

The child theme's function registers just fine. Then, the *parent* theme's `functions.php` registers a function by the same name.

PHP *really* doesn't like this. Here's a real-world example, using a Twenty Fifteen function and a child theme of Twenty Fifteen:



```
( ! ) Fatal error: Cannot redeclare twentyfifteen_widgets_init() (previously declared in
C:\wamp\www\WPShout\wp-content\themes\fifteenchild\functions.php:7) in C:\wamp\www\WPShout\wp-
content\themes\twentyfifteen\functions.php on line 141
Call Stack
# Time    Memory  Function                                                      Location
1 0.0010  243144  {main}( )                                                     ..\index.php:0
2 0.0016  246216  require( 'C:\wamp\www\WPShout\wp-blog-header.php' )           ..\index.php:17
3 0.0024  264504  require_once( 'C:\wamp\www\WPShout\wp-load.php' )             ..\wp-blog-header.php:12
4 0.0033  276392  require_once( 'C:\wamp\www\WPShout\wp-config.php' )           ..\wp-load.php:37
5 0.0050  375568  require_once( 'C:\wamp\www\WPShout\wp-settings.php' )         ..\wp-config.php:92
```

Trying to declare a Twenty Fifteen function, `twentyfifteen_widgets_init()`, in the child theme's `functions.php`

So if the parent theme isn't doing `function_exists()` checks, you need to *avoid* duplicate function names at all costs.

This also means that any custom PHP functionality the parent theme adds is simply bound to execute. If you want to change or remove pieces of that functionality, you'll have to do so manually, one step at a time, by writing your own functions that `dequeue` unwanted stylesheets, and so on.

What We've Learned

We now know the basics of child themes, and how and why to use them. To change existing themes—for example, commercial themes on client sites—you'll be using child themes constantly, so this is a really great piece of knowledge to have under your belt.

Summary Limerick

When child themes are born from their parent,
The family ties are apparent:
What they don't override
It will swiftly provide,
So inheritance comes in inherent.