Don't Fear the Custom Theme

How to build a custom WordPress theme with only four files

Linn Øyen Farley

drollic.ca

Linn Øyen Farley

- **Web designer & developer**

- Building things on the internet since 2005

- Usually working solo, or as a developer with a single designer

- Using WordPress for most client sites

I ♥ WordPress

- Amazing community and resources

- Active plugin community (don't have to reinvent the wheel)

- Super easy to hand off to clients

But

- "WordPress outputs bloated code"

- "All WordPress sites look the same"

and/or

- "I don't have time to learn PHP"

- "I tried customizing a [commercial theme provider] theme once and I couldn't find the right files to edit"

This talk will

- be a file-by-file guide to creating a fully-functional WordPress theme, based on an existing HTML/CSS design

- give an overview of the bare minimum PHP functions needed to build a WordPress theme (plus some extra stuff)

- suggest how you could use a custom WordPress theme as a tool for rapid prototyping

- use WordPress.org's Theme Guidelines[1] as a starting point for best practices

[1] developer.wordpress.org/themes/release/theme-review-guidelines/

This talk will not

- cover HTML/CSS/design principles

- 100% accurately represent how I build themes for clients

- focus on making a WordPress.org-repository-ready theme and/or incorporate every use case into a single theme

- be a hands-on workshop — I only have 45 minutes, so I'll be moving quickly

All of my slides and example files are available at drollic.ca/wcmtl16, so you can work through them at your own pace after the conference.

Please feel free to shoot me an email if you have any questions!

File structure

Theme for commercial use

Theme for commercial use

Full disclosure: I've installed this exact theme on a client's site.

There is definitely a place for commercial and child themes in the WordPress ecosystem, but creating a custom theme may be appropriate more often than you think.

Theme for a client

- 404.php
- comments.php
- footer.php
- functions.php
- header.php
- images/
  - logo.png
  - logo.svg
- index.php
- js/
  - lteie9.min.js
  - script.min.js
- page.php
- screenshot.png
- searchform.php
- single.php
- style.css

Theme for WordPress.org

- index.php
- screenshot.png
- style.css

The *recommended* theme guidelines[1] include a few more files, but the three above are the only *required* files for a theme to work properly.[2]

[1] make.wordpress.org/themes/handbook/guidelines/theme-check/#recommended
[2] developer.wordpress.org/themes/release/required-theme-files/

Theme for WordPress.org

- index.php
- screenshot.png
- style.css

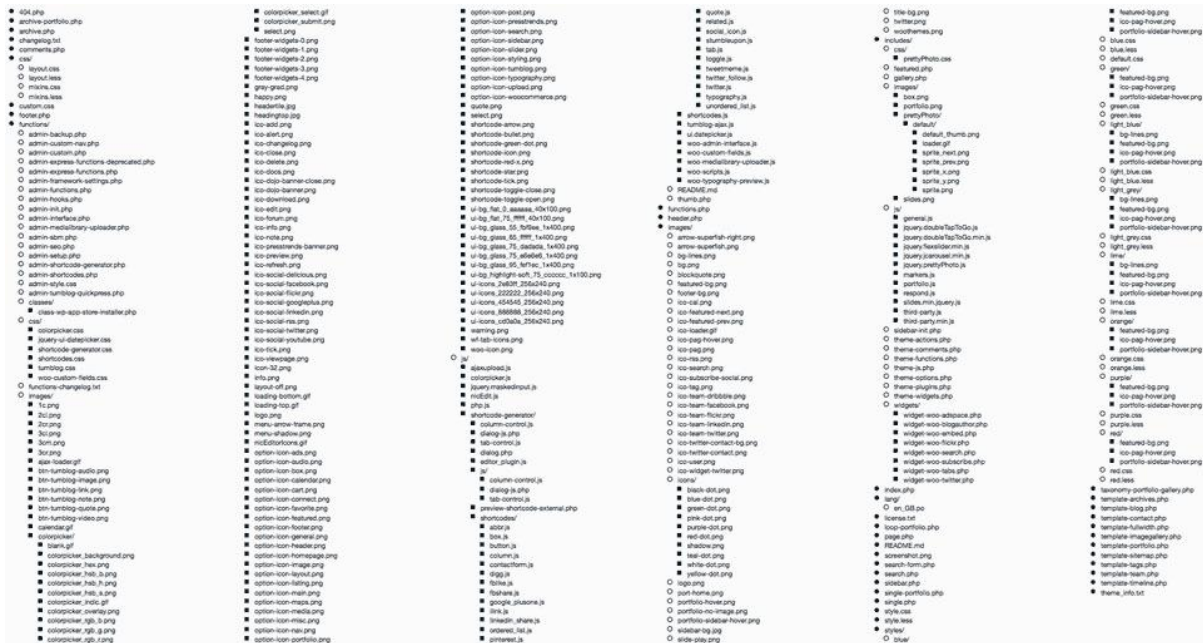...plus we'll talk about functions.php, because it's my favourite

Confession: I've ignored comments.php

Although comments.php is listed as a required file[1] on WordPress.org, you can actually get away with omitting it in your custom theme. The site will fall back to the default comments display if it doesn't find a comments.php file.
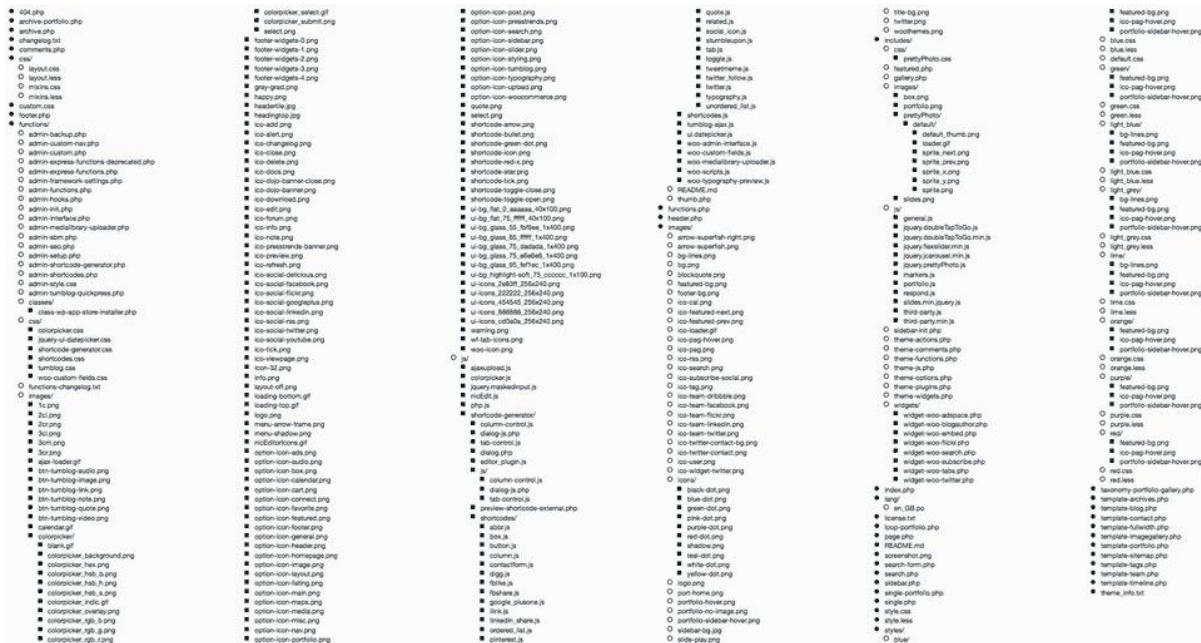
For client sites, I'd recommend including comments.php just in case – you can always copy the file from one of the default themes (like Twenty Sixteen) rather than writing your own.

[1] developer.wordpress.org/themes/release/required-theme-files/

Wait a minute

How can this →

Wait a minute

How can this →

successfully be reduced to this:

- index.php
- screenshot.png
- style.css

(you might be wondering)

The template hierarchy
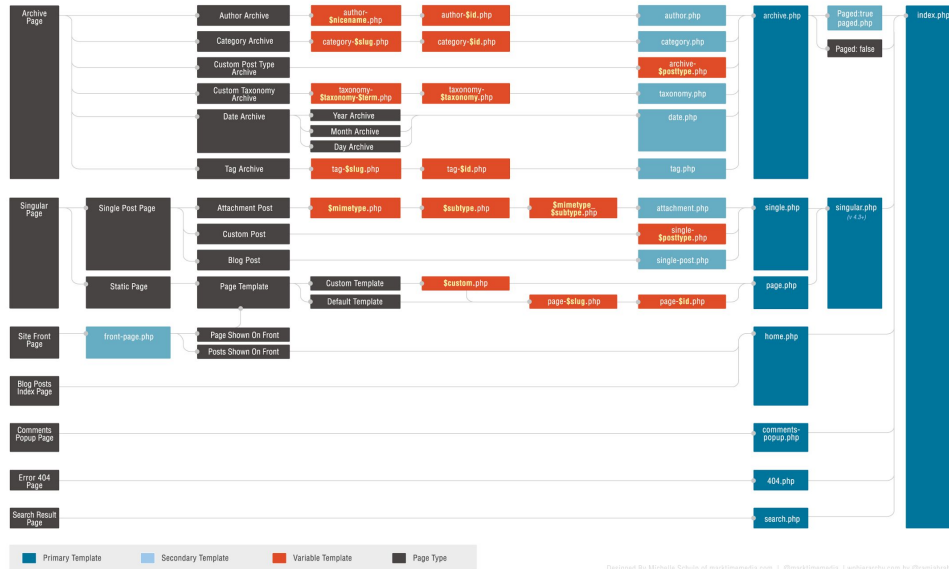
Reference: developer.wordpress.org/themes/basics/template-hierarchy/

The template hierarchy

The most important part of that giant diagram is on its right-hand side: all lines lead to **index.php**.

If your theme contains other files on the diagram, they will trump index.php, but if you *only* have index.php, that's fine too!

index.php

Hello, Hypothetical Client

List of requested features

- Responsive design

- Homepage with intro section and latest post

- About page with image gallery

- Blog with widgets in the sidebar

WordPress admin tasks

Pages → Add New

- *Home* (the intro paragraph)

- *About* (the image gallery)

- *Blog* (with no content)

- Delete *Sample Page*

Posts → Add New

- Create a few blog posts

- Delete *Hello world!*

Settings → Reading

- Choose "Front page displays: A static page"
  - Front page: *Home*
  - Posts page: *Blog*

Appearance → Menus

- Create a main navigation menu with the pages you've just made

File #1:
style.css

Start with a design

Once you have a HTML and CSS design ready to go, the first step is to rename your CSS file **style.css**, and add this comment[1] to the top:

```
/*
Theme Name: Feline Design Co.
Description: Custom theme for Feline Design Co.
Author: Linn Oyen Farley
Author URI: http://drollic.ca
Version: 1.0
*/
```

The only required line is your theme's name, but why not give yourself some credit too?

[1] codex.wordpress.org/Theme_Development#Theme_Stylesheet

Include WordPress-specific classes

WordPress auto-generates a few classes,[1] so you should account for them in your CSS to ensure they're styled properly:

Image alignment classes:

```
.aligncenter
.alignleft
.alignright
```

Image caption classes:

```
.wp-caption
.wp-caption-text
.gallery-caption
```

Accessibility classes:

```
.screen-reader-text
```

Optional post/comment classes:

```
.sticky
.bypostauthor
```

Optional menu classes:

```
.current-menu-item
.current-menu-ancestor
```

[1] make.wordpress.org/themes/handbook/guidelines/theme-check/#wordpress-generated-css-classes

File #2:
index.php

Move from HTML to PHP

Take your **index.html** file...

```
<!-- HTML goes here -->
```

...and save it as **index.php**.

```
<!-- HTML goes here -->
<?php // and also PHP ?>
```

That's all you need to do to start writing PHP in the file. Most WordPress theme files are a lot of HTML and a bit of PHP.

Interlude:
Intro to PHP

Things to keep in mind

- PHP isn't as forgiving as HTML or CSS

- Mistakes in your code = white screen of death + (sometimes) error messages that may or may not tell you where the error is

- You only need to know a little PHP syntax to build WordPress themes

- Use a code editor with syntax checking, or run your code through a validator[1] to help you find errors

[1] phpcodechecker.com

Always

- enclose PHP with opening `<?php` and closing `?>` tags

- decide to use *either* single or double quotes for consistency

- mind your semicolons

```php
<?php ?>
<?php echo "Hello world"; ?>
<?php echo 'Hello world'; ?>
<?php // this is a comment ?>
```

Reference: php.net/manual/en/

Conditionals

```php
if ( condition1 ) {
  // thing #1
} elseif ( condition2 ) {
  // thing #2
} else {
  // thing #3
}
```

← if condition #1 is true, do thing #1

← otherwise, if condition #2 is true, do thing #2

← or, if none are true, do thing #3

← don't forget the closing curly brace!

Reference: Conditional statements php.net/manual/en/language.control-structures.php

Operators

`&&`   two ampersands: *and*

`||`   two pipes: *or*

`==`   two equal signs: *is equal to*

`!=`   exclamation point and equal sign: *is not equal to*

References:
Logical operators php.net/manual/en/language.operators.logical.php
Comparison operators php.net/manual/en/language.operators.comparison.php

Loops

```
while ( condition1 == true ) {
  // do stuff
}
```

The `while` loop starts with a condition, and then specifies what to do *as long as* that condition is true.

For example, the main posts loop in WordPress states that *as long as there are posts to show* (`while` you have posts) set up each post (so you can grab its title, content, etc).

Reference: php.net/manual/en/control-structures.while.php

Variables

$ indicates a variable, either one that you want to store or one that has previously been stored.

```
$myNewVariable = 'This variable should contain some text!';
```

`$myNewVariable` now contains that string of text, which you can use later.

Reference: php.net/manual/en/language.variables.basics.php

Variables

`echo` is used to display/output the contents of an existing variable.

This:

```
<p><?php echo $myNewVariable; ?></p>
```

would output your text string on the page like this:

```
This variable should contain some text!
```

Reference: php.net/manual/en/function.echo.php

Conditional tags

WordPress has lots of its own functions that you can use in your theme's conditional statements.

We'll be talking about these ones:

`is_front_page()` ← true if you're viewing the front page

`is_page()` ← true if you're viewing a single page

`is_single()` ← true if you're viewing a single post

Reference: codex.wordpress.org/Conditional_Tags

Use unique function names

Whenever you write custom PHP functions (you'll be doing this later, in **functions.php**), it's important to start the function name with a unique string to avoid conflicts with other themes, plugins, or WordPress itself. The easiest thing to do is start them with your theme name.

This is why all of the custom functions here will start with `felinedesignco`, e.g.

```
function felinedesignco_widgets_init()
```

```
function felinedesignco_styles_and_scripts()
```

Back to file #2:
index.php

Zip that theme

If you want to see your theme in action as you work on it, it's safe to zip up your files at this point (**index.php** and **style.css**, plus any image assets) and install the theme on your site.

Be aware that the theme won't know where to find your CSS or any of your content yet, so it will look broken. Doing this now will just allow you to refresh your actual WordPress site as you edit these files and add new ones.

I wouldn't recommend doing this on a publicly accessible site!

Don't be a cowboy

From this point on, edit your theme files via FTP, *not* in Appearance → Editor.

If you make a mistake in your PHP, you may not be able to access the WordPress admin area at all to fix it. Editing via FTP lets you undo any mistakes.

wp_title()

Replace the contents of `<title></title>` with `wp_title()`. This will output a separator and the title of the page or post being viewed.

Note that if you're viewing the front page, it won't output anything.

Reference: developer.wordpress.org/reference/functions/wp_title/

wp_title() in action

Replacing this:

```
<title>Feline Design Co.</title>
```

with this:

```
<title><?php wp_title(); ?></title>
```

outputs this (when viewing the blog post titled *Blog post #1*):

```
» Blog post #1
```

Reference: developer.wordpress.org/reference/functions/wp_title/

wp_title() & bloginfo('name')

You can customize the title function by adding the site's name (the one you specify under Settings → General), and changing the separator. This:

```
<title><?php wp_title('&mdash;', true, 'right'); bloginfo('name'); ?></title>
```

changes the separator to a long dash, moves it to the right-hand side of the post/page title, and adds the site name at the end. It will output this (when viewing that same blog post):

```
Blog post #1 — Feline Design Co.
```

References:
developer.wordpress.org/reference/functions/wp_title/
developer.wordpress.org/reference/functions/bloginfo/

wp_head() & wp_footer()

These two tags must be included in your theme. They go immediately before your closing `</head>` tag and your closing `</body>` tag, respectively:

```
  <?php wp_head(); ?>
</head>
<body>
```

```
    <?php wp_footer(); ?>
  </body>
</html>
```

References:
developer.wordpress.org/reference/functions/wp_head/
developer.wordpress.org/reference/functions/wp_footer/

language_attributes() & bloginfo('charset')

You can dynamically specify the site's language and character set using these functions. This:

```
<html <?php language_attributes(); ?>>
  <head>
    <meta charset="<?php bloginfo('charset'); ?>" />
```

outputs something like this (depending on your site settings):

```
<html lang="en-CA">
  <head>
    <meta charset="UTF-8" />
```

Reference: developer.wordpress.org/reference/functions/language_attributes/

get_template_directory_uri()

Replace all relative links to design elements (such as the logo in the header area, and your stylesheet) with dynamic links to those elements in your theme folder.

This function only returns a value, so you need to echo it as well:

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

Reference: developer.wordpress.org/reference/functions/get_template_directory_uri/

get_template_directory_uri() in action

Replacing this:

```
<img src="images/logo.png" alt="" />
Feline Design Co.
```

with this:

```
<img src="<?php echo get_template_directory_uri(); ?>/images/logo.png" alt="" />
<?php bloginfo('name'); ?>
```

outputs this:

```
<img src="http://felinedesign.co/wp-content/themes/felinedesignco/images/logo.png"
alt="" />
Feline Design Co.
```

Reference: developer.wordpress.org/reference/functions/get_template_directory_uri/

wp_nav_menu()

Replace your hard-coded navigation menu with the WordPress native menu (the one you can create under Appearance → Menus).

Reference: developer.wordpress.org/reference/functions/wp_nav_menu/

wp_nav_menu() in action

Replacing this:

```
<ul>
  <li><a href="index.html">Home</a></li>
  <li><a href="about.html">About</a></li>
  <li><a href="blog.html">Blog</a></li>
</ul>
```

with this:

```
<?php wp_nav_menu( array('theme_location' => 'main-nav') ) ?>
```

Make a note of whatever you put as the `theme_location` here, because you'll need it later!

Reference: developer.wordpress.org/reference/functions/wp_nav_menu/

wp_nav_menu() in action

...outputs something like this:

```
<div class="menu">
  <ul>
    <li class="page_item page-item-6"><a href="http://felinedesign.co/about/">About</a></li>
    <li class="page_item page-item-12"><a href="http://felinedesign.co/blog/">Blog</a></li>
    <li class="page_item page-item-4 current_page_item"><a href="http://felinedesign.co/">Home</a></li>
  </ul>
</div>
```

Until you assign a menu to the `main-nav` theme location (we'll do this later), `wp_nav_menu()` will output an alphabetical list of your pages as seen above.

Reference: developer.wordpress.org/reference/functions/wp_nav_menu/

The WordPress loop

Replace your hard-coded content with dynamic content. Start by checking if any content exists, and display an error message if it doesn't:

```php
<?php if ( ! have_posts() ) { ?>
  <h1>Not Found</h1>
  <p>Sorry, nothing found.</p>
<?php } ?>
```

Reference: codex.wordpress.org/The_Loop

Review: anatomy of an if statement

A basic `if` statement is structured like this:

1. the word `if`
2. opening parenthesis
3. condition
4. closing parenthesis
5. opening curly brace
6. things that should happen if the condition is true
7. closing curly brace

have_posts()

```php
<?php if ( ! have_posts() ) { ?>
  <h1>Not Found</h1>
  <p>Sorry, nothing found.</p>
<?php } ?>
```

`have_posts()` is a WordPress function that checks if there is any content to display (either posts or pages).

The `!` means *not*, i.e. if there is *not* any content, do the following — in this case, display a "Not Found" heading and message.

Reference: developer.wordpress.org/reference/functions/have_posts/

The WordPress loop

If there *is* content available, however, you'll want to display it:

```php
<?php if ( ! have_posts() ) { // If there is no content ?>
  <h1>Not Found</h1>
  <p>Sorry, nothing found.</p>
<?php } else {
  // Display the content here!
} ?>
```

Review: anatomy of an if/else statement

When you want different stuff to happen when your condition *is* true vs. when it is *not* true, the `if` statement needs a few extra parts after the closing curly brace:

1. the word `else`
2. opening curly brace
3. things that should happen if the condition is *not* true
4. closing curly brace

The WordPress loop

```
while ( have_posts() ) {
  the_post(); ?>
  <h1><?php the_title(); ?></h1>
  <?php the_content();
}
```

If you do have content, start a `while` loop to display it.

This states that *as long as* there is content to show, set up `the_post()`.

`the_post()` contains all of the info about a post or page in WordPress, so it's ready for you to grab and use.

Reference: developer.wordpress.org/reference/functions/the_post/

the_title()

```
<h1><?php the_title(); ?></h1>
```

`the_title()` pulls whatever you've put in the post/page title field. WordPress doesn't add any formatting to your title, so you need to wrap the PHP tag in some HTML to style it.

Reference: developer.wordpress.org/reference/functions/the_title/

the_content()

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

`the_content()` pulls everything you've put into the main content editing box in WordPress, formatting and all.

Reference: developer.wordpress.org/reference/functions/the_content/

Review: the full loop

Here is the loop in its entirety, with comments throughout:

```php
<?php // If we do not have content...
if ( ! have_posts() ) {
  // ...then show an error message: ?>
  <h1>Not Found</h1>
  <p>Sorry, nothing found.</p>
<?php // Otherwise, if we do have content...
} else {
  // ...as long as there is content to show...
  while ( have_posts() ) {
    // ...set up each piece of content so we can grab stuff from it:
    the_post(); ?>
    <h1><?php the_title(); ?></h1>
    <?php the_content();
  } // end while
} // end if ?>
```

Interlude:
Theme test drive

Pages

At this point, regular pages look almost perfect:

It might look like your gallery styles aren't being applied properly, but don't worry – we'll fix that in **functions.php**

Menus

You still need to set up a menu location, so you can use Appearance → Menus and get those menu items in the right order, but we'll take care of that in **functions.php** a bit later.

Error page

The if statement that checks for no content is functioning well:

Posts page

The blog page needs some additional info/metadata on each post, beyond just the title and content. It also still needs a widgetized sidebar:

Single posts

Single blog posts need metadata and a sidebar too, plus a section to read and leave comments:

Front page

Finally, the homepage shouldn't display `the_title()` ("Home"), and it doesn't have the latest post at the bottom yet.

Back to file #2:
index.php

is_front_page()

Once you've established that you do have content to show, check whether you're viewing the front page. This should go after you set up `the_post()`.

If you are viewing the front page, display the content *without* the title:

```php
<?php if ( is_front_page() ) {
  the_content();
} ?>
```

Reference: developer.wordpress.org/reference/functions/is_front_page/

WP_Query()

Next, to grab the latest post and display it on the front page, use the function `WP_Query()`.

First, store the result in a variable:

```php
<?php $latestPost = new WP_Query('posts_per_page=1'); ?>
```

Reference: developer.wordpress.org/reference/classes/wp_query/

have_posts(), again

Then use the `have_posts()` function you've used previously, but apply it specifically to your `$latestPost` variable. This double-checks that there is a post to show, before you add a "Latest from the blog" heading and set up `the_post()`:

```php
<?php if ( $latestPost->have_posts() ) {
  // If there is a post to show, add a title before starting the loop: ?>
  <h2>Latest from the blog...</h2>
  <?php while ( $latestPost->have_posts() ) {
    $latestPost->the_post();
  }
} ?>
```

Reference: developer.wordpress.org/reference/functions/have_posts/

the_excerpt()

If there is a post to show, use `the_title()` to display its title.

Instead of displaying the full content of the post, just display its excerpt with the function `the_excerpt()`:

```
<h3><?php the_title(); ?></h3>
<?php the_excerpt(); ?>
```

Reference: developer.wordpress.org/reference/functions/the_excerpt/

the_permalink()

`the_permalink()` will echo the post's URL/permalink, which you can use to make the post title link to the full post:

```
<h3><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h3>
```

Reference: developer.wordpress.org/reference/functions/the_permalink/

the_time()

Display the date and time the post was published, using `the_time()`. To override the date and time format from Settings → General, use the Codex's date and time formatting cheatsheet.[1] This:

```
<p class="metadata">Posted on <?php the_time('F jS Y'); ?> at <?php the_time('g:i A'); ?></p>
```

outputs this:

```
<p class="metadata">Posted on July 23rd 2016 at 3:30 PM</p>
```

Reference: developer.wordpress.org/reference/functions/the_time/
[1] codex.wordpress.org/Formatting_Date_and_Time

the_terms()

Display the post's categories using `the_terms()`. This:

```
in <?php the_terms($post->ID, 'category'); ?>
```

outputs something like this:

```
in <a href="http://felinedesign.co/category/cats/" rel="tag">Cats</a>
```

Reference: developer.wordpress.org/reference/functions/the_terms/

Review: the latest post

Here's the complete block of code that will display the post's permalink, title, metadata, and excerpt:

```
<h3><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h3>
<p class="metadata">Posted on <?php the_time('F jS Y'); ?> at <?php the_time
('g:i A'); ?> in <?php the_terms($post->ID, 'category'); ?></p>
<?php the_excerpt(); ?>
```

And this is how it looks →

Latest from the blog...

Blog post #2

Posted on July 23rd 2016 at 3:30 PM in Cats

Tom havana brown grimalkin siberian lion. Malkin americ
bombay abyssinian. Norwegian forest. Savannah siames
british shorthair for devonshire rex yet kitty, and siamese

Homepage: complete!


```php
<?php if ( is_front_page() ) {
  the_content();
  $latestPost = new WP_Query('posts_per_page=1');
  if ( $latestPost->have_posts() ) {
    // If there is a post to show, add a title before starting
the loop: ?>
    <h2>Latest from the blog...</h2>
    <?php while ( $latestPost->have_posts() ) {
      $latestPost->the_post(); ?>
      <h3><a href="<?php the_permalink(); ?>"><?php the_title
(); ?></a></h3>
      <p class="metadata">Posted on <?php the_time('F jS Y'); ?
> at <?php the_time('g:i A'); ?> in <?php the_terms($post->ID,
'category'); ?></p>
      <?php the_excerpt();
    } // end while
  } // end if
} // end if ?>
```

is_page()

You can use the same code to display post metadata on both the main blog page and on single posts. To avoid displaying metadata on pages, though, add another condition to the loop first:

```php
<?php } elseif ( is_page() ) {
    // If this is a regular page, just display the title and content: ?>
    <h1><?php the_title(); ?></h1>
    <?php the_content();
} else {
    // Otherwise, display the title and content plus the metadata ?>
    <h1><?php the_title(); ?></h1>
    <p class="metadata">Posted on <?php the_time('F jS Y'); ?> at <?php the_time('g:i A'); ?> in <?php
the_terms($post->ID, 'category'); ?></p>
    <?php the_content();
} // end if ?>
```

Reference: developer.wordpress.org/reference/functions/is_page/

comments_template()

This function adds a comment form, plus any comments that have been left on the post.

To avoid displaying comments on pages, add the function at the end of the loop, *after* you've established that you're not viewing a page:

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

Reference: developer.wordpress.org/reference/functions/comments_template/

comments_template() in action

One response to "Blog post #2"

1. _Linn_ says:
   July 23, 2016 at 3:30 pm

   This is a test comment!

   Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Obviously this could use a little styling, but it works!

next_posts_link() & previous_posts_link()

Once you have more posts than you've specified under "Blog pages show at most" (Settings → Reading), you'll need pagination links to navigate the blog.

Add the following code between `the_content()` and `comments_template()`:

```
<p class="navigation">
    <?php next_posts_link('&larr; Older posts');
    previous_posts_link('Newer posts &rarr;'); ?>
</p>
```

References:
developer.wordpress.org/reference/functions/next_posts_link/
developer.wordpress.org/reference/functions/previous_posts_link/

Pagination links in action

Depending on where you are in the blog, these links may look like any of the following examples:

berian and bobcat siamese. British shorth abby. Tomcat lynx. British shorthair russiar btail. Lynx malkin, yet birman, and burmes siamese. Manx grimalkin so bobcat so lior kitty. Munchkin tabby yet jaguar for lynx.

← Older posts   ← Older posts   Newer posts →   Newer posts →

Save space for widgets

You'll be creating a widgetized area in **functions.php**, so save some space for it now.

Before you start the loop, make sure you're not on a page, and then open the primary `<section>`:

```php
<?php if ( ! is_page() ) { ?>
  <section class="primary">
<?php }
while ( have_posts() ) { ?>
```

Save space for widgets

Then just before closing the main `<section>`, check to make sure you're not on a page again, close the primary `<section>` and add an `<aside>`:

```php
<?php if ( ! is_page() ) { ?>
  </section><!-- .primary --><aside class="secondary">
    <?php // widget area will go here ?>
  </aside><!-- .secondary -->
<?php } ?>
</div><!-- .container -->
</section><!-- .main -->
```

Request the widget area

Within the `<aside>`, use the `dynamic_sidebar()` function to request the widget area called "blog-widget-area" (not yet created, you'll do this in **functions.php**):

```
<aside class="secondary">
    <?php dynamic_sidebar('blog-widget-area'); ?>
</aside><!-- .secondary -->
```

Make a note of the `dynamic_sidebar` name you're requesting here! In this case it's "blog-widget-area".

Reference: developer.wordpress.org/reference/functions/dynamic_sidebar/

dynamic_sidebar() in action

For now, this will give you an empty sidebar on the posts page and on single posts:

is_single()

Post titles should display differently on single posts vs. on the main posts page.

Inside the final `else` statement, add one more conditional statement to take care of this:

```php
<?php // Otherwise, display the title and content plus the metadata
if ( is_single() ) {
  // If you're viewing a single post, display the title as h1: ?>
  <h1><?php the_title(); ?></h1>
<?php } else {
  // Otherwise, display the title as h2 and link it to the full post: ?>
  <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>
<?php } ?>
<p class="metadata">Posted on <?php the_time('F jS Y'); ?> at <?php the_time('g:i A'); ?> in <?php the_terms
($post->ID, 'category'); ?></p>
```

Reference: developer.wordpress.org/reference/functions/is_single/

File #3:
functions.php

Not required, but super useful

This file is basically a plugin bundled with your theme. Keep in mind that all of its added functionality will disappear if you switch your theme!

There are tons of things you can do with a functions file. I don't have a lot of time left, so I'm just going to cover a few examples of what's possible.

Create your functions file

Make a new file called **functions.php**, and add opening and closing PHP tags:

```php
<?php
?>
```

register_nav_menus()

Remember making a note of the theme location "main-nav" when you were replacing your hard-coded menu with `wp_nav_menu()`?

It's time to create that location, so you can assign a menu to it under Appearance → Menus:

```php
// Register menu(s)
register_nav_menus(
  array('main-nav' => 'Main Navigation')
);
```

Reference: developer.wordpress.org/reference/functions/register_nav_menus/

register_nav_menus()

If your theme is going to use more than one menu, you can register them all in one go:

```php
// Register menu(s)
register_nav_menus(
  array(
    'main-nav' => 'Main Navigation',
    'footer-nav' => 'Footer Navigation'
  )
);
```

To display the menu assigned to Footer Navigation, you would use this code:

```php
<?php wp_nav_menu( array('theme_location' => 'footer-nav') ); ?>
```

register_nav_menus() in action

Much better!

Note: After adding the code, you need to go to Appearance → Menus in the WordPress admin area, create a menu, and select *Main Navigation* as its "Theme location" for this to work.

register_sidebar()

Although the word "sidebar" is right there, this lets you create any widgetized area (that you can then drag widgets into, under Appearance → Widgets).

```php
// Register widgetized area(s)
function felinedesignco_widgets_init() {
  register_sidebar( array(
      'id' => 'blog-widget-area',
      'name' => 'Blog Widget Area',
      'description' => 'Appears on the blog and single posts.',
      'before_title' => '<h3 class="widget-title">',
      'after_title' => '</h3>',
      'before_widget' => '<section class="blog-widget">',
      'after_widget' => '</section><!-- .blog-widget -->',
  ) );
}
add_action('widgets_init', 'felinedesignco_widgets_init');
```

Reference: developer.wordpress.org/reference/functions/register_sidebar/
Generator that will write the code for you: generatewp.com/sidebar/

register_sidebar()

Remember "blog-widget-area"? Whatever you asked for with `dynamic_sidebar()` in **index.php**:

```php
<?php dynamic_sidebar('blog-widget-area'); ?>
```

needs to match whatever you put as the `id` in **functions.php**:

```php
'id' => 'blog-widget-area'
```

register_sidebar()

You can register several widget areas in one go with this function, and then display each one using `dynamic_sidebar('the-id-you-chose')`:

```php
// Register widgetized area(s)
function felinedesignco_widgets_init() {
  register_sidebar( array(
    'id' => 'blog-widget-area',
    'name' => 'Blog Widget Area',
    'description' => 'Appears on the blog and single posts.',
    'before_title' => '<h3 class="widget-title">',
    'after_title' => '</h3>',
    'before_widget' => '<section class="blog-widget">',
    'after_widget' => '</section><!-- .blog-widget -->',
  ) );
  register_sidebar( array(
    'id' => 'footer-widget-area',
    'name' => 'Footer Widget Area',
    'description' => 'Appears in the footer.',
    'before_title' => '<h4 class="widget-title">',
    'after_title' => '</h4>',
    'before_widget' => '<section class="footer-widget">',
    'after_widget' => '</section><!-- .footer-widget -->',
  ) );
}
add_action('widgets_init', 'felinedesignco_widgets_init');
```

register_sidebar() in action

We have a sidebar!

Note: After adding the code, you need to go to Appearance → Widgets in the WordPress admin area, and drag some widgets into the area called *Blog Widget Area* for this to work.

wp_enqueue_style()

Instead of hard-coding stylesheets into the `<head>` of your site:

```
<link rel="stylesheet" href="<?php echo get_template_directory_uri(); ?>/style.css" type="text/css"/>
<link href='https://fonts.googleapis.com/css?family=Bitter' rel='stylesheet' type='text/css'>
```

enqueue them in **functions.php**:

```php
// Register and enqueue styles and scripts
function felinedesignco_styles_and_scripts() {
  wp_enqueue_style('core', get_stylesheet_uri());
  wp_enqueue_style('fonts', 'https://fonts.googleapis.com/css?family=Bitter');
}
add_action('wp_enqueue_scripts', 'felinedesignco_styles_and_scripts');
```

Reference: developer.wordpress.org/reference/functions/wp_enqueue_style/

wp_enqueue_style()

```
wp_enqueue_style('core', get_stylesheet_uri());
```

This enqueues the theme's default stylesheet. `get_stylesheet_uri()` will grab the URL of **style.css** in the theme folder.

```
wp_enqueue_style('fonts', 'https://fonts.googleapis.com/css?family=Bitter');
```

This enqueues a Google Fonts stylesheet. It's located off-site, so you need to give the full URL.

Reference: developer.wordpress.org/reference/functions/wp_enqueue_style/

wp_enqueue_script()

Within `felinedesignco_styles_and_scripts()`, you can use `wp_enqueue_script()` to enqueue scripts, if your theme has any. To enqueue a JavaScript file called **global.js**, for example, you would add this below your `wp_enqueue_style()` lines:

```
wp_enqueue_script('global', get_template_directory_uri().'/js/global.js', array
('jquery'), '1.0', true);
```

This enqueues a file called **global.js** located in a folder called "js" inside the main theme folder. The file depends on jQuery to function (so it should load after jQuery), it's at version 1.0, and it should appear just before the closing `</body>` tag instead of in the `<head>`.

Reference: developer.wordpress.org/reference/functions/wp_enqueue_script/

Conditionally enqueuing scripts

WordPress comes with lots of JavaScript files ready to go, but until you enqueue them they won't be loaded in your theme files.

comment-reply.js is one of these pre-registered scripts. It enhances threaded comments on single posts, but you only want it to load when it's useful. To limit when this file is included, add this conditional statement to the end of your `felinedesignco_styles_and_scripts()` function:

```
if ( is_singular() && get_option('thread_comments') && comments_open() ) {
  wp_enqueue_script('comment-reply');
}
```

Reference: developer.wordpress.org/reference/functions/wp_enqueue_script/#defaults

Review: enqueuing styles and scripts

Here is the full block of code for your **functions.php** file:

```php
// Register and enqueue styles and scripts
function felinedesignco_styles_and_scripts() {
  // Load stylesheets:
  wp_enqueue_style('core', get_stylesheet_uri());
  wp_enqueue_style('fonts', 'https://fonts.googleapis.com/css?family=Bitter');

  // Only include the line below if you actually have a file called global.js:
  wp_enqueue_script('global', get_template_directory_uri().'/js/global.js', array('jquery'),
'1.0', true);

  // Conditionally load threaded comments script:
  if ( is_singular() && get_option('thread_comments') && comments_open() ) {
    wp_enqueue_script('comment-reply');
  }
}
add_action('wp_enqueue_scripts', 'felinedesignco_styles_and_scripts');
```

excerpt_more

By default, WordPress adds […] to the end of `the_excerpt()`. To replace this with an ellipsis and a "Continue reading [post title] →" link to the full post, include this code in **functions.php**:

```php
// Append ellipsis and continue reading link to automatic excerpts
function felinedesignco_excerpt( $more ) {
  return ' &hellip; <a href="'.get_permalink().'">Continue reading &ldquo;'.get_the_title().'&rdquo; &rarr;</a>';
}
add_filter('excerpt_more', 'felinedesignco_excerpt');
```

Reference: developer.wordpress.org/reference/hooks/excerpt_more/

excerpt_more in action

This:

avannah. Birman persian scottish fold so jaguar tal
persian so tabby for bengal for savannah lion. Oc
Abyssinian […]

now looks like this:

avannah. Birman persian scottish fold so jaguar tal
persian so tabby for bengal for savannah lion. Oc
Abyssinian … Continue reading "Blog post #2" →

add_image_size()

To make WordPress automatically generate more image sizes than the three available under Settings → Media, add this code to **functions.php**:

```
// Register custom image sizes
add_image_size('hero', 1090, 320, true); // cropped to exactly 1090x320 pixels
add_image_size('narrow', 150, 9999, false); // sized to 150 pixels wide by
proportional height (up to 9999 pixels tall)
```

If you uploaded images to your Media Library before adding this code, you now need to run Regenerate Thumbnails.[1] All future uploads will generate your custom image sizes automatically, so you only need to run the plugin once.

Reference: developer.wordpress.org/reference/functions/add_image_size/
[1] wordpress.org/plugins/regenerate-thumbnails/

image_size_names_choose

This code will make your new image sizes available when using Add Media:

```php
// Add custom sizes to the WordPress Media Library
function felinedesignco_choose_sizes( $sizes ) {
  return array_merge( $sizes, array(
    'hero' => __('Hero'),
    'narrow' => __('Narrow')
  ) );
}
add_filter('image_size_names_choose', 'felinedesignco_choose_sizes');
```

Make sure the lowercase names above match whatever you called your new image sizes in the `add_image_size()` function.

Reference: developer.wordpress.org/reference/hooks/image_size_names_choose/

Remove inline [gallery] styles

The default WordPress gallery inserts some inline CSS that you may want to override in your theme. You could use lots of `!important`s in your stylesheet, but it's easy to just stop the styles from loading entirely:

```
// Remove inline WordPress gallery styles
add_filter('use_default_gallery_style', '__return_false');
```

Reference: developer.wordpress.org/reference/hooks/use_default_gallery_style/

Remove inline [gallery] styles

This changes this:

to this:

Now the border, alignment, and width rules from **style.css** are being applied to the gallery items, instead of being overridden by the inline styles.

add_theme_support('post-thumbnails')

Enable featured images for posts and pages by adding this to **functions.php**:

```php
// Add support for featured images
add_theme_support('post-thumbnails');
```

Then in **index.php**, check for and display the featured image if there is one. This code will get the size called "thumbnail", and add the class `alignright`:

```php
if ( has_post_thumbnail() ) {
  the_post_thumbnail('thumbnail', array('class' => 'alignright'));
}
```

References:
codex.wordpress.org/Post_Thumbnails
developer.wordpress.org/reference/functions/the_post_thumbnail/

File #4:
screenshot.png

Take a screenshot

With the theme ready to go, simply screenshot the actual website to take care of your final required file, **screenshot.png**.

- Maximum size: 1200x900px

- The screenshot "should be of the actual theme as it appears with default options, not a logo or mockup"[1]

[1] make.wordpress.org/themes/handbook/review/required/#screenshot

The screenshot in action

Hello again, Hypothetical Client

List of requested features, revisited

✓ Responsive design

List of requested features, revisited

✓ Homepage with intro section and latest post

List of requested features, revisited

✓ About page with image gallery

List of requested features, revisited

✓ Blog with widgets in the sidebar

Questions?

@LinnOyenFarley
linn@drollic.ca

drollic.ca/wcmtl16

Slides, HTML & CSS template, finished theme