VCD-3600

WEB DESIGN II

TIPS & TRICKS: Wordpress Themes, Part 2

Introduction

After setting up the files for your theme as described in Part 1 the next task we have to complete is to enable content to come from Wordpress rather than being hard-coded into the page. This will vary for every particular site and page structure, so what follows must be understood as a general model and not a line-for-line specification.

Displaying a Feed of Posts

WordPress centers around creating posts that are published in blog-like style. This typically assumes that the site will have two particular page templates:

- A home page that contains a feed of all posts sorted from my recent to oldest, and
- A page on which a single post is displayed.

Later we'll discuss how to set these up as separate pages, but suffice to know that you can acheive both through your index.php file using the following approach. First, let's image we're in the setting described by the first bullet, where we want a feed of all posts to appear.

1. Remove all but one of your sample posts from the static content you have in place on this page. You should have a single sample post remaining, perhaps organized in a `<div>`.
2. Right before the start of the one remaining sample post, add the following lines:

```
<?php if (have_posts()) :?>
    <?php while (have_posts()) : the_post(); ?>
```

In breif, this little bit of WordPress code checks for any posts on the first line, and if there are posts in the system, the next line will be processed. In that second line, a loop is kicked off that will repeat for each post in the system.

3. Right underneath the end of the one remaining sample post, add the following lines:

```
    <?php endwhile; ?>
<?php else : ?>
    <!-- Backup content goes here as needed. -->
<?php endif; ?>
```

This bit of code finished off the loop, then provides an "otherwise" clause that allows you to provide backup content that would appear if no posts are found in the system.

Now you just need to replace static content in page with PHP tags that bring in specific chunks of content from WordPress for each post. Refer to the following and add each as needed:

- `the_author` – displays the author's name. Alternatively, you can use `the_author_meta('__value__')` to display a particular peice of data about the author, depending on what you provide for `'__value__'`. You can choose from options such as:
  - » `'first_name'` – first name of the post's author.
  - » `'last_name'` – last name of the post's author.

  There are many other options described on this page from the WordPress Codex. Note that a simpler alternative is to just call `the_author()`.
- `the_category(', ')` – displays a list of any and all categories in which this post is organized, separated by a comma and a space. Note that you can change the contents of the string parameter to be whatever delimiter you'd like.
- `the_content()` – displays the content of the post.
- `the_date()` – displays the date the post was published, by default, using this format:
  January 1, 2016
  This can be altered by passing a special character set in as the first parameter. More information about this function can be found here and more information about date characters can be found here.
- `the_ID()` – displays the unique ID that WordPress generated for the post.
- `the_permalink()` – displays the URL of the post.
- `the_title()` – displays the title of the post.
- `comments_number()` – displays the number of comments a post currently. Specific parameters can be provided in order to configure what the system should display when there are no comments, when there is 1 comment, and when there is more than 1 comments. See this page from the WordPress Codex for more information.
- `post_class()` – displays the class attribute WordPress generated for the post.

Put all this together based on your needs and you can create a neatlyformatted feed of posts. Here is an example of how these could be used together:

```
<?php if (have_posts()) : ?>
    <?php while (have_posts()) : the_post(); ?>
    <article <?php post_class(); ?>>
        <h2><a href="<?php the_permalink() ?>"><?php the_title();?></a></h2>
        <p>Published on <?php the_date() ?>
           by <?php the_author(); ?>
           in <?php the_category(", ") ?></p>
        <div class="entry-content">
            <?php the_content();?>
        </div>
        <p><a href="<?php the_permalink() ?>"><?php comments_number(); ?></a></p>
    </article>
    <?php endwhile; ?>
<?php endif; ?>
```

Controlling Loops

It is very common to build off of the idea of a blog to create customized sites that show off content from one or more categories of posts. Think of a news website. The home page itself might feature some recent news from across the site but then have several sections where top articles from different categories are listed and organized. Furthermore, the site might also have different sections that show only content from a certain category, such as "Sports" or "Local News." In order to accomplish lists of articles like this we can use a lot of the same features as were covered above, but with more fine-tuned control over the specific set of posts we're showing in a certain area.

Working with query_posts()

One very powerful tool available to us is the query_posts() method, which allows us to filter a request for WordPress content based on parameters we pass into this function. Here are some options for the parameters:

- `cat` – the id number for a category that would result in filtering the set of posts to those from the provided category.
- `category_name` – similar to the previous, this is the "slug" for a category that results in filtering the set of posts to those from the provided category.
- `posts_per_page` – indicates how many posts to retrieve "per page." You can use this to determine how many posts get returned, or for more advanced features such as paginating a list of posts.
- `offset` – indicates how many posts to skip from the beginning of the list before beginning to display. This may be helpful if you have a setup where one post is displayed first as a "featured" post and then followed by a more mundane list of posts. In this case, you'd run to separate calls to `query_posts()`. The first one you'd set `posts_per_page` to `1` and then in the second call you'd set `offset` to `1`. See the exampes that follow for more information.
- `order` – sets whether to retrieve posts in ascending order with `'ASC'` or descending order with `'DESC'`. Since the default funcationality is to resemble a blog which sorts by post date with the most recent first, `'DESC'` is the default.
- `orderby` – sets what field to order posts by. The default is to order by the post date. A full list of other options can be found here.

Example 1: Posts from Two Categories

This snippet shows one way you could display two separate lists of posts from two specific categories, such as on a home page of a news website:

```
<!-- First we get the most recent 5 posts from the "news" category. -->
<div class="news-posts">
    <h2>News</h2>
    <?php
        $args = array(
            'category_name' => 'news',
            'posts_per_page' => 5
        );
        query_posts($args);
    ?>
    <?php if (have_posts) : ?>
    ... [loop code goes here] ...
    <?php endif; ?>
</div>
<!-- Next we get the most recent 5 posts from the "sports" category. -->
<div class="sports-posts">
    <h2>Sports</h2>
    <?php
        $args = array(
            'category_name' => 'sports',
            'posts_per_page' => 5
        );
        query_posts($args);
    ?>
    <?php if (have_posts) : ?>
    ... [loop code goes here] ...
    <?php endif; ?>
</div>
```

Example 2: Featured Post with Simple List Following

This snippet demonstrates how query_posts can be used to query the same category several times in order to treat one set of posts differently from another set.

```
<div class="sports">
    <!-- First we retrieve the first item from the Sports category to "feature" -->
    <div class="featured">
        <?php
            $args = array(
                'category_name' => 'sports',
                'posts_per_page' => 1
            );
            query_posts($args);
        ?>
        <?php if (have_posts()) : ?>
            <?php while (have_posts()) : the_post(); ?>
            <!-- We'll show more details for this first post than for others... -->
            <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>
            <p>Written by <?php the_author(); ?> on <?php the_date(); ?></p>
            <?php the_content(); ?>
            <?php endwhile; ?>
        <?php endif; ?>
    </div>
```

```
<!-- Next we retrieve the next 5 items and show them in a short list. -->
<ul>
    <?php
        $args = array(
            'category_name' => 'sports',
            'posts_per_page' => 5,
            'offset' => 1
        );
        query_posts($args);
    ?>
    <?php if (have_posts()) : ?>
        <?php while (have_posts()) : the_post(); ?>
        <li><a href="<?php the_permalink() ?>"><?php the_title();?></a></li>
        <?php endwhile; ?>
    <?php endif; ?>
</ul>
```

Maintaining the Default Query

As convenient as query_posts() is one thing to note is that it does cancel the existing query WordPress has generated. On the home page this might not be a bad thing. But on other pages such as category.php we will benefit from leaving more of WordPress's hard work in place. To do so, use the following base snippet for your query:

```
<?php
    global $wp_query;
    $custom_args = array(
        'posts_per_page' => 5,
        'offset' => 1
    );
    $args = array_merge($wp_query->query_vars, $custom_args);
    query_posts($args);
?>
<!-- Loop code goes here (if, while, etc.) ... -->
```

This ensures that the original query created by WordPress carries forward and that your custom arguments simply override existing ones, rather than starting from scratch with just your custom arguments.