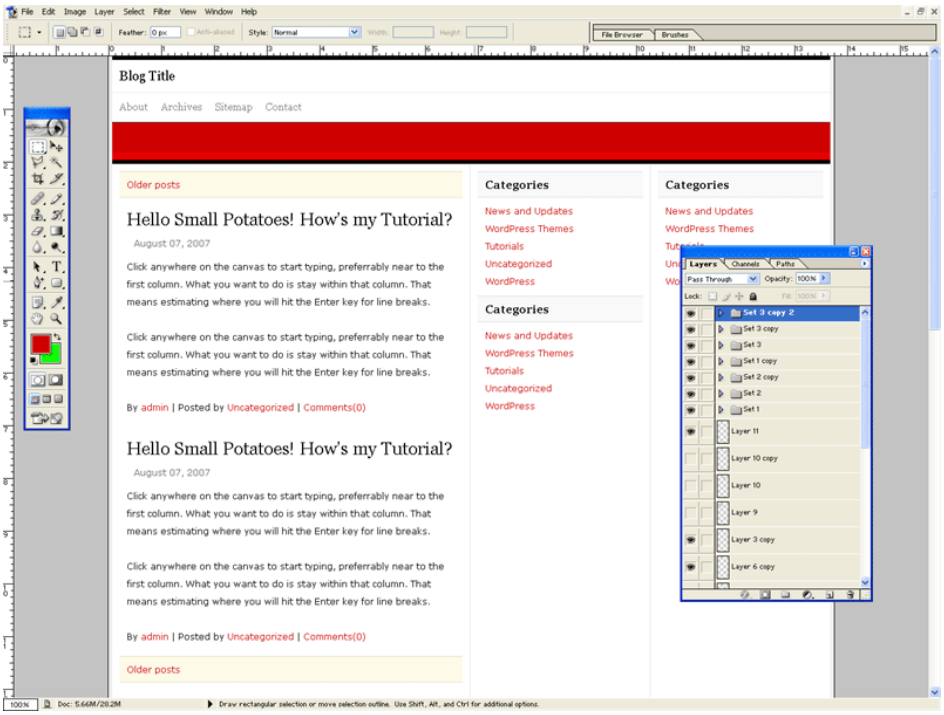
# How to create a WordPress theme from scratch

by Small Potato of Wpdesigner.com




## Part 3 - How to Code a WordPress Theme

If anything happens to you or your computer because you decided to install all the tools, suggested by this tutorial, to create a WordPress theme, I cannot be held responsible for it.

In part one and two, you learned how to design your theme and how to slice it. The design in the screenshot above is what you should have ended up with. Unlike part one and two, part three is NOT optional. You must read it.

Before you can start building your WordPress theme:

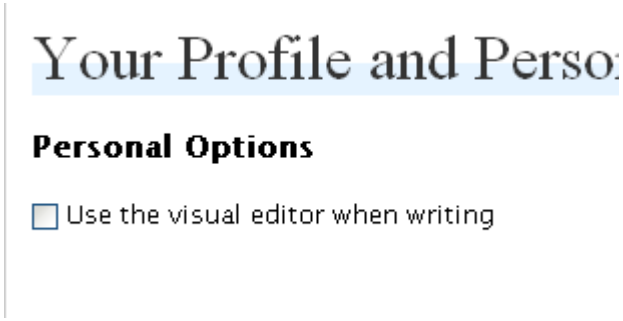
- **Download WordPress**
- **Install WordPress on your computer using Xampp or Mamp**. You really need to insall it on your computer for easy access, fixing, and testing. It's important to note that I'm using Windows XP and Xampp so the instructions below are based on those tools.
- **Grab a text editor.** For Window users, **Notepad** will be your text editor.
- **Install Firefox**
- **Install the Web Developer add-on for Firefox.** Or, use the XHTML and CSS validtors.

---

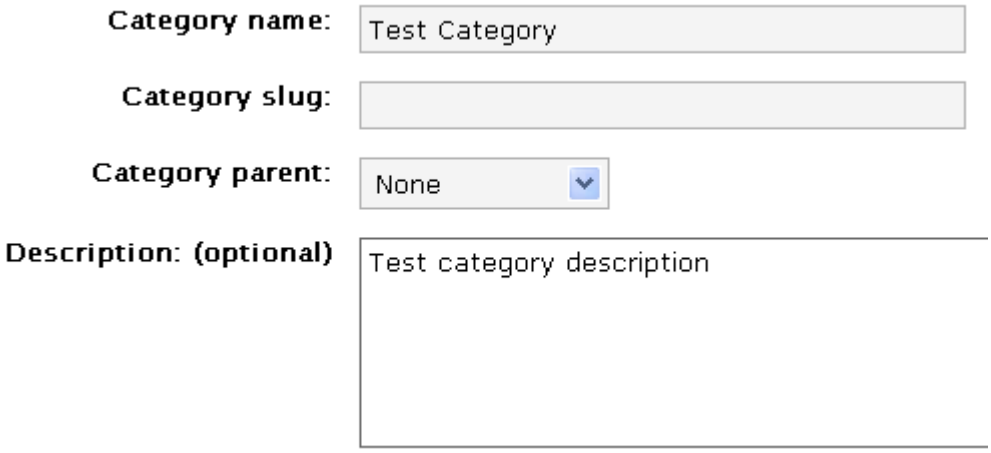
# Configuring Your Local WordPress Install

After logging in:

- Go to **Users > Your Profile**
- Uncheck "**Use the visual editor when writing**"

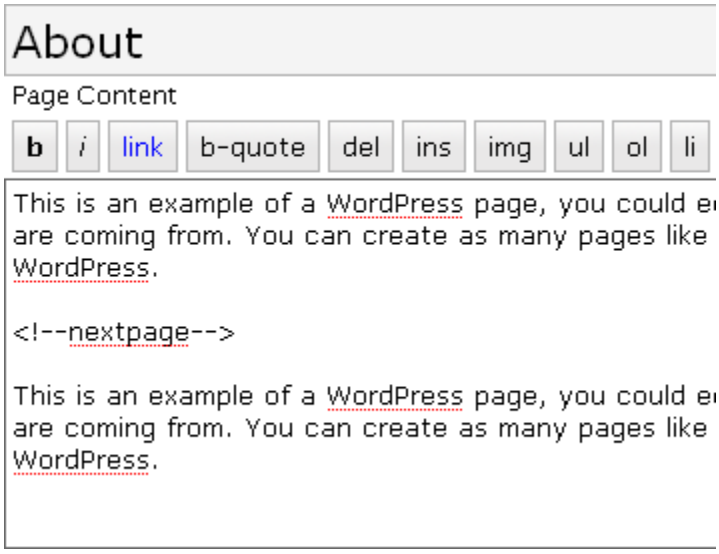



- Also, change your password to something you can remember.
- Click **Update Profile**
- Go to **Manage > Categories**
- Add a new category. Fill in **Category Name** and **Description**. Leave everything else empty. Name the category anything you want and description can be, "blah blah blah."



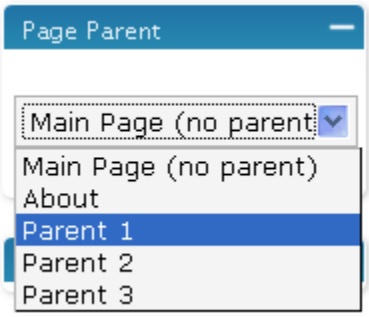

- Click **Add category**
- Go to **Manage > Pages**. Edit the **About** page.

**Editing About page**. Type **<!-- nextpage-->** under the default paragraph that it gives you for the **About** page. Copy the paragraph and paste it under **<!--nextpage-->**. Click Save.




Go to **Write > Write Page** to create a new page. Make sure you're on **Write Page** not **Write Post**. Use "**Parent 1**" for page title. Leave everything else empty and click **Publish**. Create two more pages, **Parent 2** and **Parent 3**.

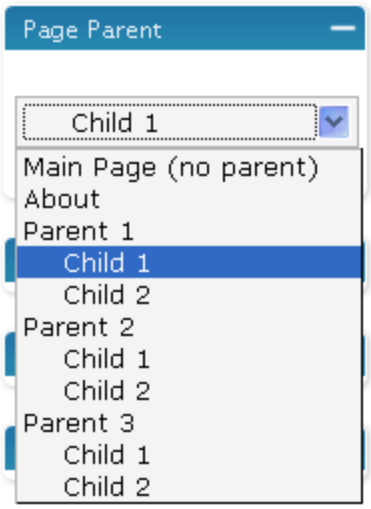
While you're at it, create a new page named **Child 1**. Instead of publishing it right away, go to the right hand side and select its parent page. Select the **Parent 1** page. Now publish it.



Create five more child-pages. One more for **Parent 1**, two child-pages for **Parent 2** and **3**. Here's what you should have:

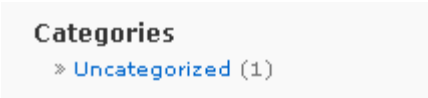
| | |
|---|---|
| 2 | About |
| 3 | Parent 1 |
| 6 | — Child 1 |
| 7 | — Child 2 |
| 4 | Parent 2 |
| 8 | — Child 1 |
| 9 | — Child 2 |
| 5 | Parent 3 |
| 10 | — Child 1 |
| 11 | — Child 2 |

Create a grandchild page. Name it **Grand Child 1**.  This time, do it for **Child 1 of Parent 1** only.

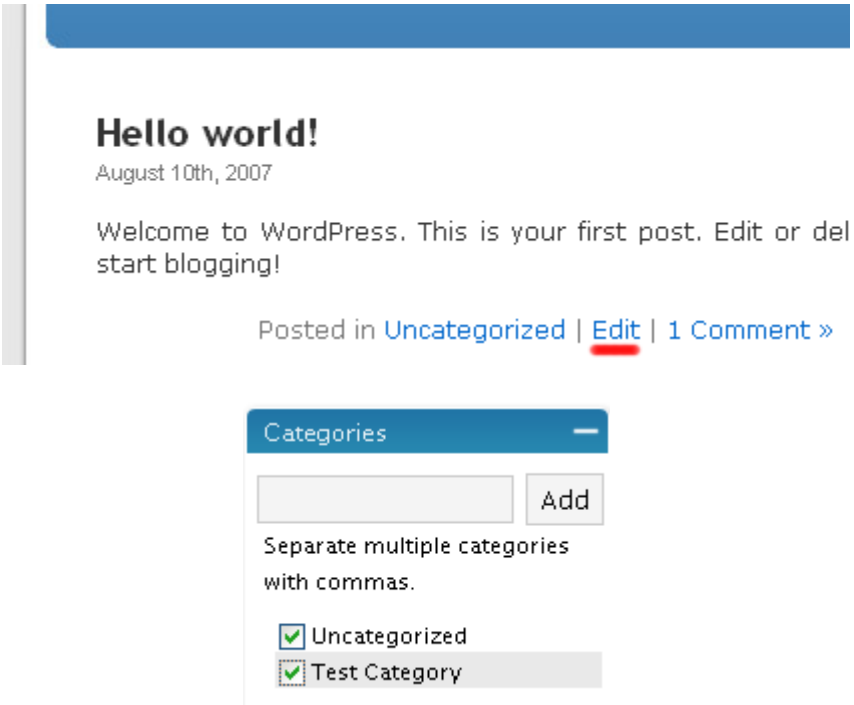


Let's go back to the front page to see what you've created.


How come the **Categories** area lists only one category? The other categories are not listed because you haven't posted anything under them.




Let's edit the **Hello World** post to file it under the other categories too. After editing, click **Save**.



Go back to the front page to see the change. Now both checked categories are listed.


This is the end of basic blog congfiguration. Moving on to...

## Making Dummy Posts for Testing Your Theme

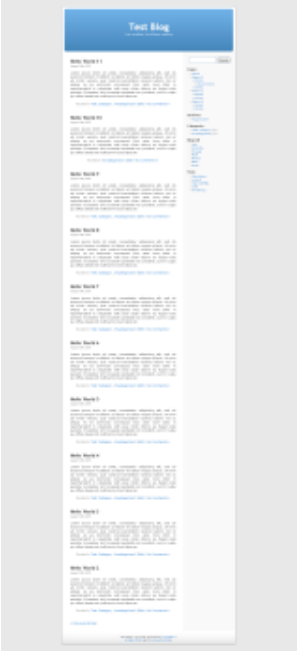
For each post that you create, check or file it under both categories. **You need to create 20 posts**. Simply go to **Write > Write Post**. Fill in a title and some texts for that blog post. Here's what I use for every dummy post:

"**Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim...**"

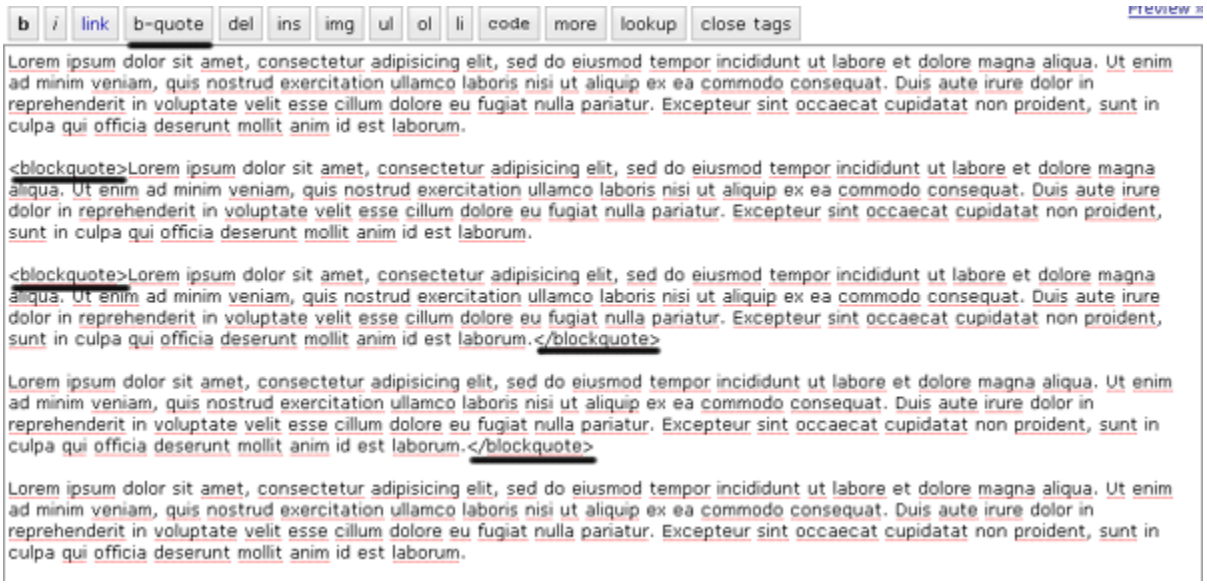
The paragraph above is what most designers use to lay out fake content so they can test their designs. I've included the paragraph above in a file named **lorem.txt**. Check your tutorial folder for it.

You can have 20 posts that say the same message, but each of the last 10 posts have to be unique and they must follow certain formats, which I'll show you. For now, go create your first 10 posts and then come back for instructions on the last 10 unique posts.

**Are you finished with the first ten posts? You sure?** Here's how my test blog looks after the first 10.

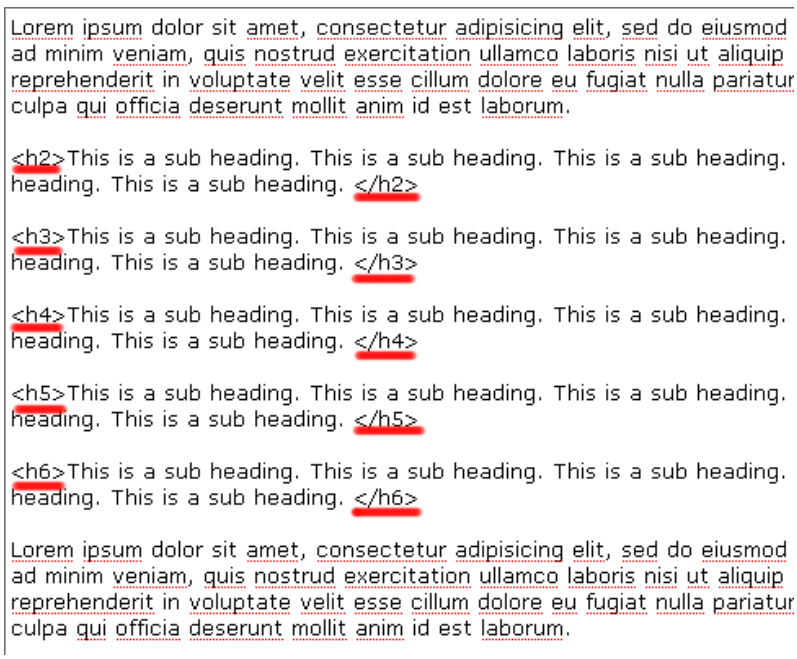


**Unique Post #1**: To put a paragraph within a blockquote, highlight the entire paragraph, then click on the **b-quote** button that I've highlighted below. You need to use a blockquote within a blockquote. In the screenshot below, notice how one set of blockquote codes are nesting within another set? Also, type or paste one paragraph before the blockquotes and one paragraph after the blockquotes.

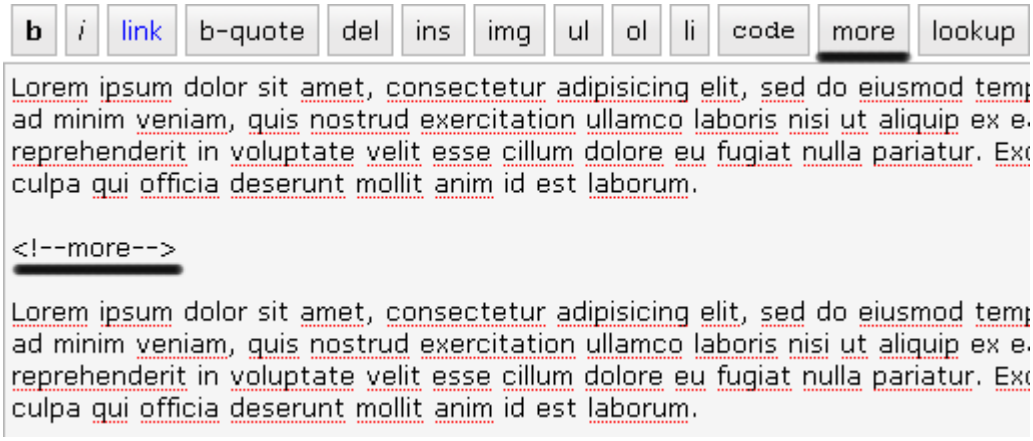



**Unique Post #2**: This one needs a lot of sub-headings. Sub-headings are H2, H3, H4, H5, and H6. H1 is the main heading, but no one should use that within a blog post so you're not going to use it in this step.

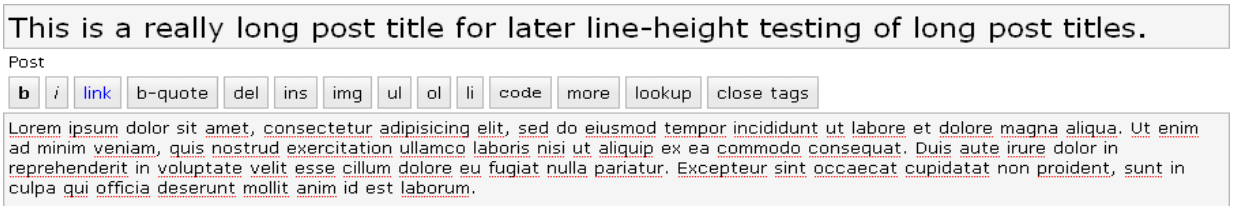
**Notice**: the opening of a code is within < >, but the closing has an extra forward slash like </ >.



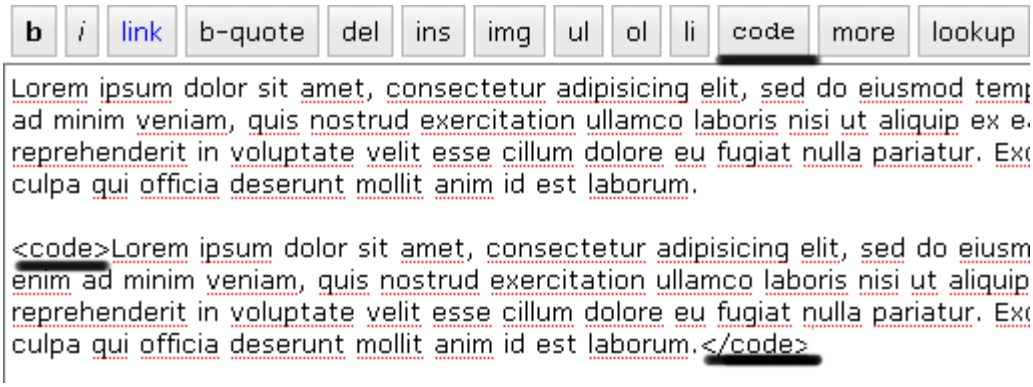
**Unique Post #3**: Use **<!--more-->**.




**Unique Post #4**: Use a really long post title.



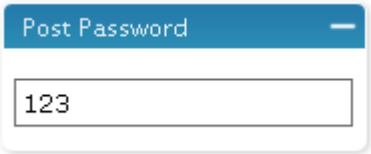

**Unique Post #5**: Use the code button.



**Unique Post #6**: Like editing the **About** page, but use **<!--nextpage-->** for a post this time.

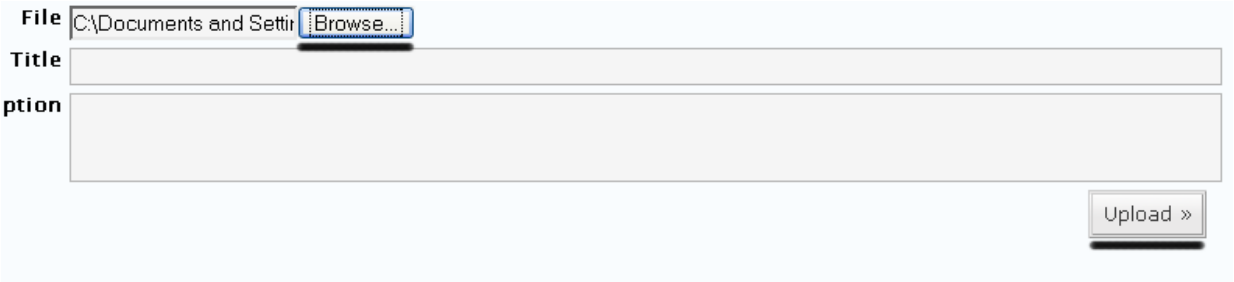
> Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod temp
> ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex e
> reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Ex
> culpa qui officia deserunt mollit anim id est laborum.
>
> <!--nextpage-->
>
> Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod temp
> ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex e
> reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Ex
> culpa qui officia deserunt mollit anim id est laborum.

**Unique Post #7**: You don't need to do anything special. Simply password protect it:

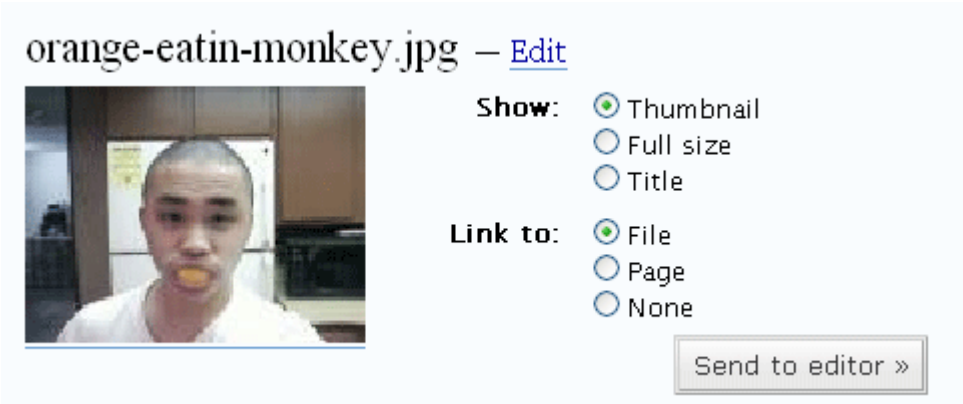



**Unique Post #8**: Upload some images through the WordPress media uploader and use those images within your post. This one is a little complicated...

First, you need to upload the photo / image:



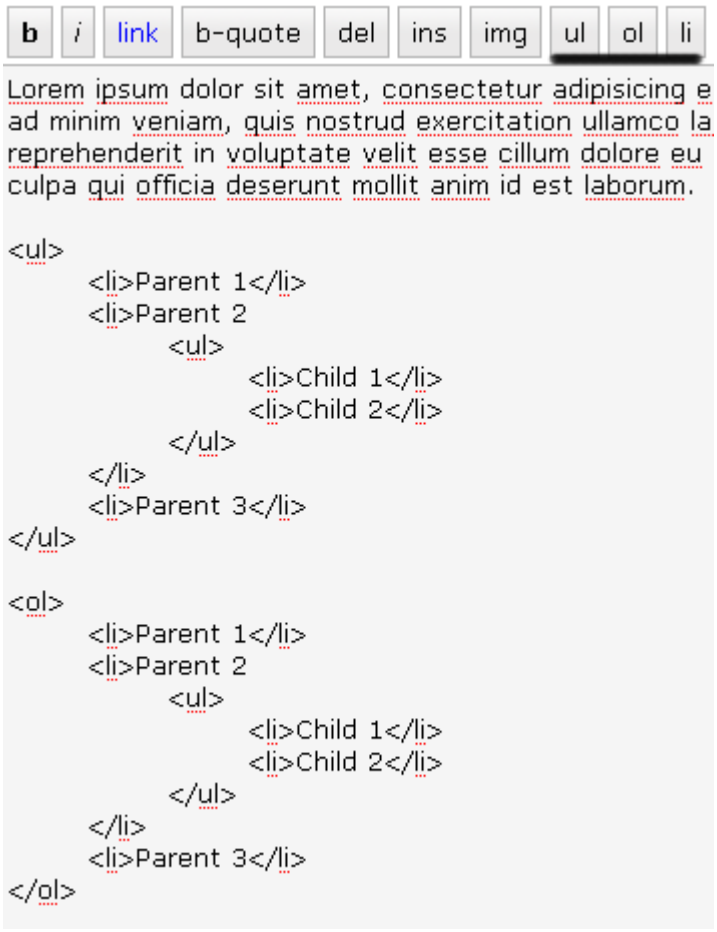
Next, I added my photo three times to one blog post.




Then I added **class='alignleft'** to the first image code, **class='alignright'** to the second image code, and **class='centered'** to the third image code.

```
<a href='http://localhost/test/wp-content/uploads/2007/08/orange-eatin-monkey.jpg' title='orange-eatin-monkey.jpg'><img
src='http://localhost/test/wp-content/uploads/2007/08/orange-eatin-monkey.thumbnail.jpg' alt='orange-eatin-monkey.jpg'
class='alignleft' /></a> Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute
irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

<a href='http://localhost/test/wp-content/uploads/2007/08/orange-eatin-monkey.jpg' title='orange-eatin-monkey.jpg'><img
src='http://localhost/test/wp-content/uploads/2007/08/orange-eatin-monkey.thumbnail.jpg' alt='orange-eatin-monkey.jpg'
class='alignright' /></a>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute
irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

<a href='http://localhost/test/wp-content/uploads/2007/08/orange-eatin-monkey.jpg' title='orange-eatin-monkey.jpg'><img
src='http://localhost/test/wp-content/uploads/2007/08/orange-eatin-monkey.thumbnail.jpg' alt='orange-eatin-monkey.jpg'
class='centered' /></a>
```

**Unique Post #9**: Use ordered and unordered list codes. Don't worry, it's easy.

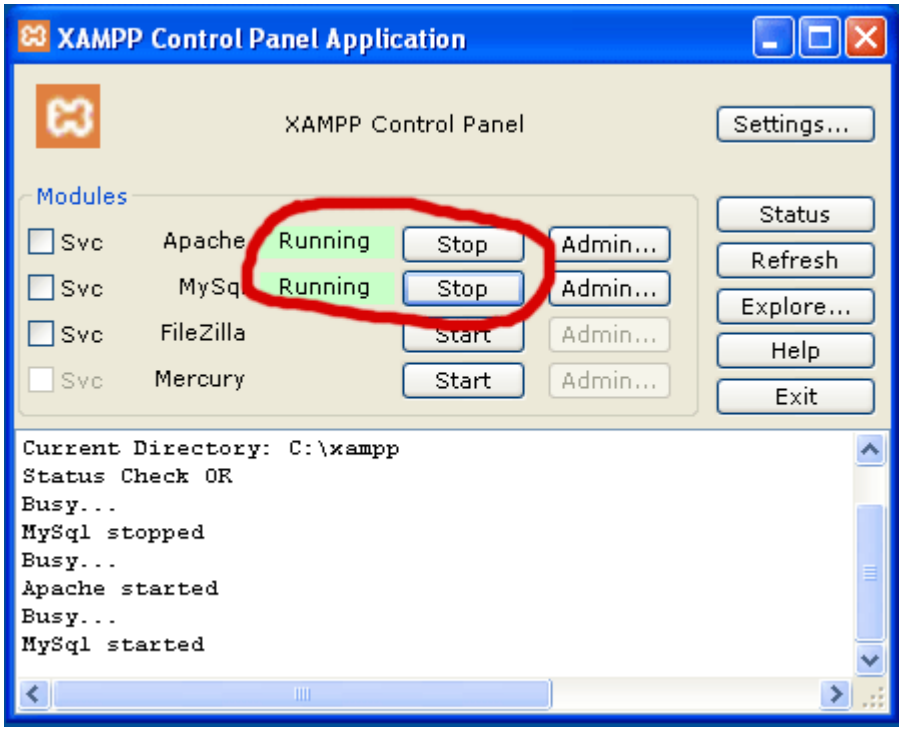



**Unique Post #10**: This one is really hard.... Just kidding, you can do whatever you want with this one. You actually needed to create only 9 unique posts. I wish I could see your face when you read, “this one is really hard.” I kid! I kid!

Are we all done with configurations and setting up dummy posts? Yes, we are, but there's one more thing. Open a folder, any folder. At the top of the page, go to **Tools > Folder Options**. Click on the **View** tab. Uncheck the box next to **Hide extensions for known file types**.

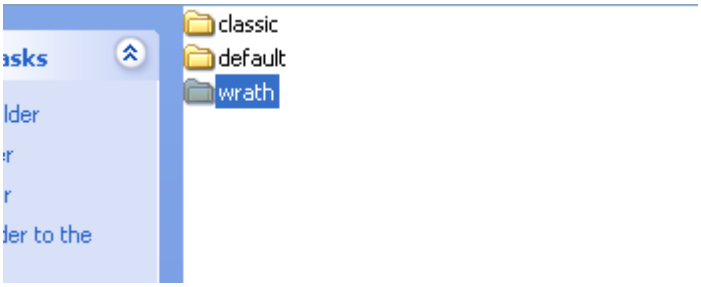
# Creating the style.css and index.php files

Open **Xampp Control Panel**. Navigate to your xampp folder. Usually **My Computer > Local Disc > xampp**. Double click on **xampp-control.exe**. A window will pop up. Click on **Start** to turn on **Apache** and **MySQ**L.




You may minimize or exit this window now. Don't worry about exiting the window by pressing the X button. Xampp won't close itself unless you click on the word "**Exit**."

Go to your WordPress themes folder and create a new folder named "**wrath**." Your themes folder is at **My Computer > Local Disc > xampp > htdocs > wordpress > wp-content > themes**.



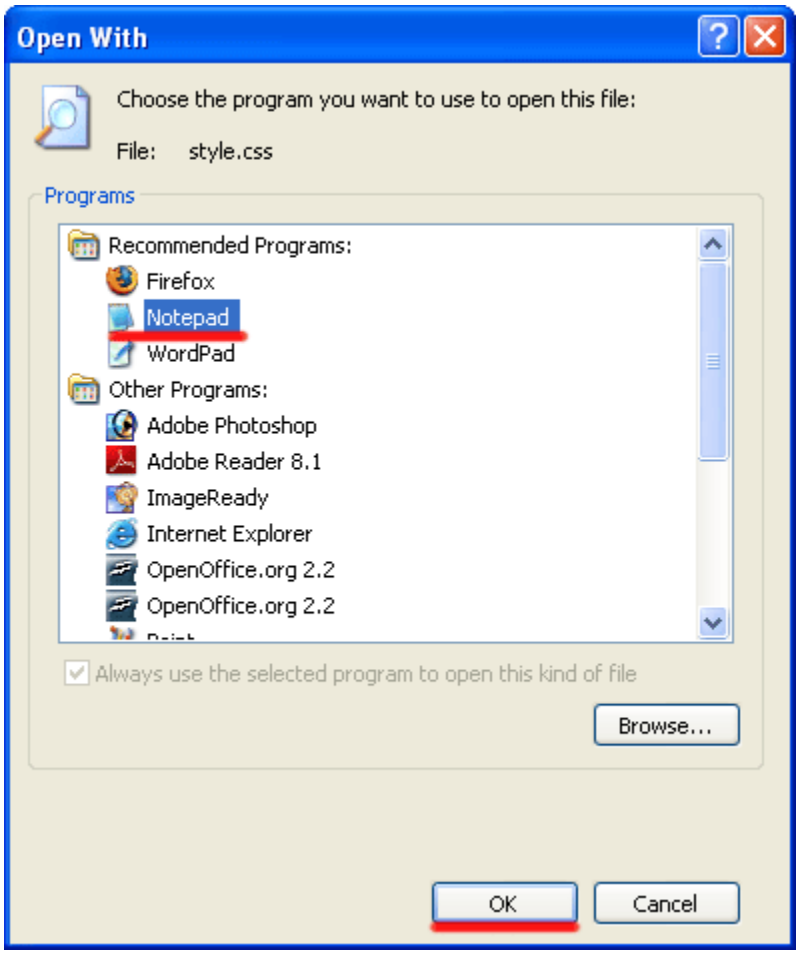

Everything related to the theme you'll be working on is going to go into the "**wrath**" folder.

---

## Creating the style.css file

Open up the "**wrath**" folder, right click within its window and select **New > Text Document**. Change the name of the new document to **style.css**.


After renaming, right click on that file and select **Properties**. When you see the Properties window, look for **Open With**, then click the change button. Select **Notepad** and click **OK**.




Now you can view and edit the **style.css** file using Notepad. Next, create another text document. Rename it to **index.php**. **Index.php** tells your blog where everything goes and **style.css** controls how everything looks. Do the **Properties > Open With** step for the **index.php** file. After that, you should be able to open **style.css** and **index.php** using Notepad.

---

# Add theme information to style.css

Open up **style.css** using Notepad, type the following:

```
/*
Theme Name: Wrath
Theme URI: http://www.wpdesigner.com/
Description: Light version of Wrath, used for WordPress tutorial.
Version: 1.0
Author: Small Potato
Author URI: http://www.wpdesigner.com/
*/
```

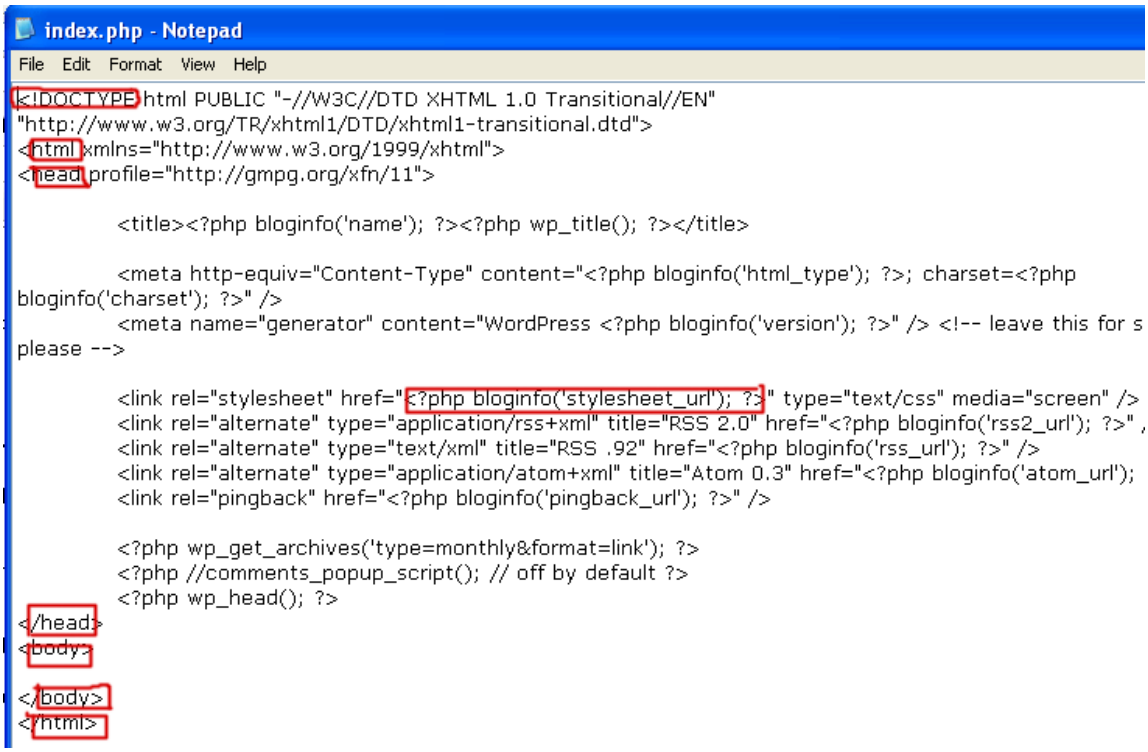
**File > Save** your style.css file, then close it. Here's what you should see on the **Presentation** page of the WordPress administration panel:


Click on the **Wrath 1.0** link or the empty thumbnail. When you go back to the front page, you should see a blank page.

Put **style.css** aside for now to start working on **index.php**. You need to know where everything goes before you can pretty it up with **style.css**.

# Starting the index.php file

Open the tutorial folder and find **index-001.txt**. Copy everything in there to your **index.php** file.




There's a reason why I let you copy and paste certain codes instead of copy and pasting everything. That reason is you need to actually type the codes to learn them. However, you can copy and paste the complicated parts that I can't explain to you right away.

In the image above, I circled the codes that you should focus on.

**Doctype** - Indicates what kind of codes you're using to code your theme. Doctype is not important at this point. I'm pointing out **Doctype** so you don't have think about it. **The rest of the codes are basically saying:**

**<html>** is where my web page starts.

**<head>** is where the head of my web page starts. Every web page has a head and a body. **</head>** is where the head ends.

---

**<?php bloginfo('stylesheet_url'); ?>** is a PHP function that calls for the location of the **style.css** file so my theme can link to it and style everything on my pages. Anytime codes are wrapped in **<?php** and **?>**, it's PHP and it's different from the rest of my codes. In PHP, **<?php** is start and **?>** is end.

So:

- **<?php** - start PHP
- **bloginfo('stylesheet_url')** - call for the location of style.css
- **;** - stop calling for **style.css**. The semicolon is one way of closing a set of codes within PHP.
- **?>** - end PHP

**<body>** is where the body starts. The body is everything that I see and read on a web page.

**</body>** is where the body ends.

**</html>** is where my web page ends. Nothing comes after that.

# The Loop: Calling for content

In between **<body>** and **</body>**, type:

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

<?php endwhile; ?>

<?php endif; ?>

**What just happened?**

- **if(have_posts())** - checks to see **if** you **have** any **post**.
- **while(have_posts())** - if you do have it, **while** you **have** any **post**, execute the_post().
- **the_post()** - call for the posts to be displayed.
- **endwhile;** - close **while()**
- **endif;** - close **if()**

In WordPress, what you just typed is called **The Loop**. It's one of the most important sets of codes to know, if not the most important.

Save **index.php**. Go to your blog's home page and refresh it. You should see a blank page. What? Don't freak out on me. You put in **The Loop** to call for blog posts, but you didn't tell **The Loop** what to display. Let's add **<?php the_content(); ?>** to it like this:

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

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

<?php endwhile; ?>

<?php endif; ?>

---

Save your file and refesh your home page again. You should now be able to see only the content of your blog posts. But, what about the blog title? Ah, I'm glad you asked.

Here comes **<?php the_title(); ?>** to the rescue. Add it to **The Loop**, above **the_content** like so:

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

**<?php the_title(); ?>**

<?php the_content(); ?>

<?php endwhile; ?>

<?php endif; ?>

Save **index.php** and refresh the blog's home page. You spot the pattern yet? Every time you make a change, save your file and then go back to refresh the page to see the change. Can you have the date on each post to let users know when each was posted? Sure you can.

Add **<?php the_time('F jS, Y') ?>** to **The Loop**, under **the_title**:

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

<?php the_title(); ?>

**<?php the_time('F jS, Y'); ?>**

<?php the_content(); ?>

<?php endwhile; ?>

<?php endif; ?>

Save **index.php** and refresh the home page. That's nice and all, but you want to separate the title and date? Ok, but walk before you run. Let's deal with all the PHP mumbo jumbo first.

Besides the_content, the_title, and the_titme, what else is missing? How about the author, categories, and the number of comments?

Under **the_content**, add :

**<?php the_author(); ?>** for identity of the author.

**<?php the_category(', ') ?>** for the categories the post was filed under.

<?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?> for the number of comments.

The three separate sets of codes above, together, they are called **post meta data** or simply **entry meta**.

---

**You should have**:

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

<?php the_title(); ?>

<?php the_time('F jS, Y'); ?>

<?php the_content(); ?>

<?php the_author(); ?> <?php the_category(', ') ?> <?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?>

<?php endwhile; ?>

<?php endif; ?>

Save **index.php** and refresh the home page. Now you have the entry meta line, but it doesn't really make sense. Let's add some words in front of each entry meta area.

**Posted by** <?php the_author(); ?> **under** <?php the_category(', ') ?> **with** <?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?>

**Save** and **refresh**.

Ok, now that you have the content, title, date, author, list of categories, and number of comments, what else is missing? Ooh I know, the navigation links, those small, but important **Previous page** and **Next page links**. Add <?php posts_nav_link(); ?> before and after the loop because you want it to appear twice only. If you haven't notice, everything inside The Loop repeats itself 10 times.

So here's how you add it:

<?php if(have_posts()) : ?>**<?php posts_nav_link(); ?>**<?php while(have_posts()) : the_post(); ?>
<?php the_title(); ?>
<?php the_time('F jS, Y'); ?>
<?php the_content(); ?>
Posted by <?php the_author(); ?> under <?php the_category(', ') ?> with <?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?>
<?php endwhile; ?>
**<?php posts_nav_link(); ?>**
<?php endif; ?>

Save and refresh. **Notice: The first posts_nav_link** appear in between **if** and **while**. The second one appear between **endwhile** and **endif**.

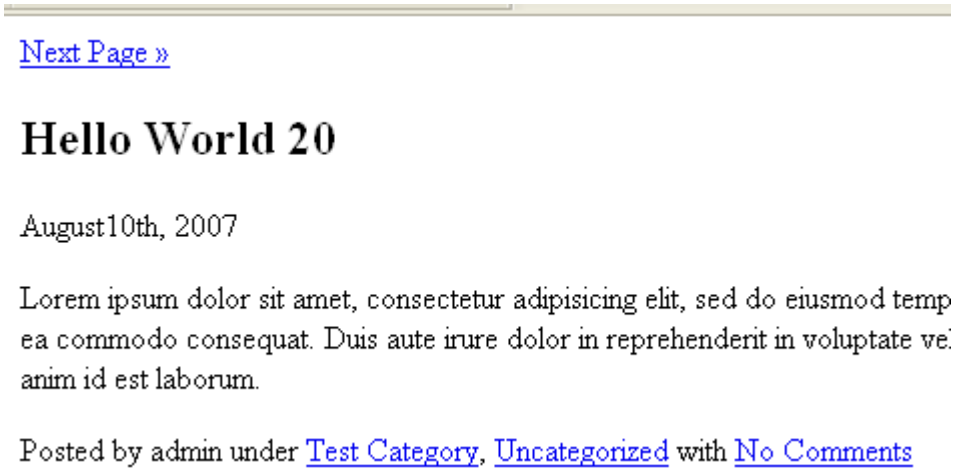
---

Now that you have those links too, what could possibly be missing? The answer is quite a few things, but you're probably bored with PHP and can't wait to start oraganizing all this stuff, understandable. Let's organize!

Now, you'll be working with XHTML and PHP at the same time. First, tackle the post titles. Add the **h2** sub-heading tags around the_title. H2 is XHTML, not PHP.

<h2><?php the_title(); ?></h2>

You could use h1, h3, h4, h5, or h6, but for the sake of this tutorial, use **h2**! Save and refresh. Just one simple set of **h2** tags, look at it now:




The **h2** tag is also useful for styling later on. For example, you can use the **style.css** file to target the **h2** tag to control the font size, color, and etc. of the post title, but let's play with that later. Since we're working on the post title, let's take it a little bit further. To turn the **post title** into a **post title link**, add a set of link tags or anchor tags around **the_title**.

<h2><a href=””><?php the_title(); ?></a></h2>

Save and refresh. As you can see, your post titles are links now, but they point to nowhere. What's missing? The value in between the quotes of **href** is missing. Let's add something that let the post title links point to the right pages. That something is **<?php the_permalink(); ?>**.

<h2><a href=”<?php the_permalink(); ?>”><?php the_title(); ?></a></h2>

Save and refresh. Now, the **post titles** point to the single pages that match each blog post. You use **the_permalink** because it calls for the right location for each blog post.

The next step is a little bit confusing. You're going to add **title=”<?php the_title(); ?>”** to the link / achor tag. First add **title=””**.

<a href=”<?php the_permalink(); ?>” **title=””**><?php the_title(); ?></a>

You know that something goes in between the quotes of **title=””**, but what? It's **the_title** again. Like this: <a href=”<?php the_permalink(); ?>” title=”**<?php the_title(); ?>**”><?php the_title(); ?></a>

Yes, you use the_title twice on the same line. One sits between the **<a>** and **</a>** tags. The other sits inside the **<a>** tag like this <a title=”**<?php the_title(); ?>**”>.

Save and refresh. Now, when you put your mouse over a **post title link**, a link title description shows up. Ok, we're finished with the **post title links**. Before we move on, here's what you should have:

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

<h2><a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>"><?php the_title(); ?
></a></h2>

<?php the_time('F jS, Y'); ?>

<?php the_content(); ?>

Posted by <?php the_author(); ?> under <?php the_category(', ') ?> with <?php
comments_popup_link('No Comments', '1 Comment', '% Comments'); ?>

<?php endwhile; ?>

<?php posts_nav_link(); ?>

<?php endif; ?>
```

Let's move on to the date. We need a way to separate it from everything else like we did with the post titles. So, we're going to use the **div** tags like this:

** **<?php the_time('F jS, Y'); ?>** **

What just happened? You wrapped a set of **div** tags around the date and name them **entry-date** by using the **class=””** attribute. What's an attribute in XHTML? It doesn't matter. Just know that when someone asks you what's **class=””**, it's an attribute.

That's it. You're finished with the date. Disappointing huh? For now, the **div** is nothing, but an invisible box that doesn't really do anything. It's just there wrapping around the date. Save and refresh.

We're going to do the same thing for **the_content**. Here it is:
** **<?php the_content(); ?>** **

**Save and refresh**. You might think that all these names: entry-date and entry-content for examples are confusing and how would a beginner know all the names and which to use.

---

Well, you don't have to use those names. You don't even have to call them entry-date and entry-content. Call them anything you want. Name them whatever you want. For examples: big potato and small potato. It doesn't matter. What matter is that they make sense to you and tells you exactly what and where they are. Instead of calling them small potato and big potato, I call them entry date and entry content because it's simply easier to name them based on what they are.

Moving on to entry-meta. Wrap a set of **div** tags around this area just like we did with the date and the content.

** **Posted by <?php the_author(); ?> under <?php the_category(', ') ?> with <?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?>** **

**Save** and **refresh**. Now, we'll add ** ** and ** ** around everythin within **The Loop** to separate them from everything outside of **The Loop**.

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

<div class="post">

<h2><a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>"><?php the_title(); ?
></a></h2>

<div class="entry-date"><?php the_time('F jS, Y'); ?></div>
<div class="entry-content"><?php the_content(); ?></div>

<div class="entry-meta">Posted by <?php the_author(); ?> under <?php the_category(', ') ?> with
<?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?></div>

</div>

<?php endwhile; ?>

<?php posts_nav_link(); ?>

<?php endif; ?>
```

At this point, the codes are cluttered and unorganized. Each time you look at it, you have to waste time on figuring out which code closes which. So, use tabs to add indents / organization to your codes like this:

```
<?php if(have_posts()) : ?>

<?php posts_nav_link(); ?>

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

<div class="post">

	<h2><a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>"><?php the_title(); ?></a></h2>
	<div class="entry-date"><?php the_time('F jS, Y'); ?></div>

	<div class="entry-content">

		<?php the_content(); ?>

	</div>

	<div class="entry-meta">Posted by <?php the_author(); ?> under <?php the_category(', ') ?> with <?php comments_popup_link('No Comments', '1 Comment', '% Comments'); ?></div>

</div>

<?php endwhile; ?>

<?php posts_nav_link(); ?>

<?php endif; ?>
```

Now, let's do something about **posts_nav_link**. Although it's now separated from everything within The Loop, we still need a way to target it to style it later on. To do that, we're going to add **div** tags around **posts_nav_link** and name it **class="navigation"**.

```
<div class="navigation"><?php posts_nav_link(); ?></div>
```

Remember to do the step above twice because we're using **posts_nav_link** in two locations.

```
<?php if(have_posts()) : ?>

<div class="navigation"><?php posts_nav_link(); ?></div>

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

<div class="post">
        <h2><a href="<?php the_permalink(); ?>" title="<
        <div class="entry-date"><?php the_time('FjS, Y');

        <div class="entry-content">

                <?php the_content(); ?>

        </div>
        <div class="entry-meta">Posted by <?php the_au
Comments'); ?></div>
</div>

<?php endwhile; ?>

<div class="navigation"><?php posts_nav_link(); ?></div>

<?php endif; ?>
```

We have the basic content and navigation covered. Now, let's add somet extra stuff to the loop.

First one is **<?php wp_link_pages(); ?>**. It's for the **<!--nextpage-->** that you added to one of the unique posts earlier. Find that post and you'll see something like **Pages: 1 2**, with 1 and 2 being links.

Second addition is **<?php edit_post_link('Edit', '<p>', '</p>'); ?>**. When you are logged in, that code will give you an edit link. That way, when you spot an error, you can simply click on the edit link to edit that certain blog post, instead of having to go into the WordPress administration page to search for it.

Both first and second additions are conditionals. They'll only appear when you they're in use. Otherwise, they'll be invisible.

After finishing the content and navigation links, we're going to work on the header area. But before that, add another set of **div** tags to separate what you have so far from the header area.

Wrap ** ** and ** ** around everything.

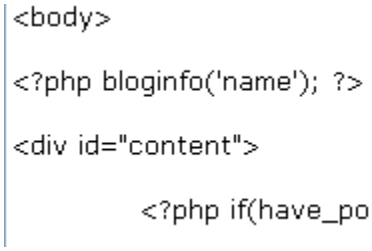
```
<div id="content">

	<?php if(have_posts()) : ?>

	<div class="navigation"><?php posts_nav_link(); ?></div>

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

	<div class="post">
		<h2><a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>"><?php
the_title(); ?></a></h2>
		<div class="entry-date"><?php the_time('F jS, Y'); ?></div>

		<div class="entry-content">

			<?php the_content(); ?>
			<?php wp_link_pages(); ?>
			<?php edit_post_link('Edit', '<p>', '</p>'); ?>

		</div>
		<div class="entry-meta">Posted by <?php the_author(); ?> under <?php the_category(',
') ?> with <?php comments_popup_link('No Comments', '1 Comment', '%
Comments'); ?></div>
	</div>

	<?php endwhile; ?>

	<div class="navigation"><?php posts_nav_link(); ?></div>

	<?php endif; ?>

</div>
```

For the step above, you used **id=”content”** instead of **class=”content”**. You can use a Class multiple times on one page, but you can't use an ID multiple times on one page. Once something is labeled **id=”content”**, that is it. Nothing else can be **id=”content”**. Class is repeatable. ID isn't repeatable. That's all you need to remember for now.

# Add Header

Above the **id=”content”** div, type:

<?php bloginfo('name'); ?>




Save and refresh. That's your blog's title. You can always change what it says at the WordPress administration panel, on the **Options** page.

Wrap **h1** tags around it:

**<h1>**<?php bloginfo('name'); ?>**</h1>**

Save and refresh. Now, your home page knows that the blog title is the main heading. Next is turning the blog title into a link, linking back to the home page.

<h1>**<a href="<?php bloginfo('url'); ?>">**<?php bloginfo('name'); ?>**</a>**</h1>

Save and refresh. If you did it right, your blog title, which is the main heading, should turn into a link that points back to the home page. You used **<?php bloginfo('url'); ?>** as the value for the link's location. **bloginfo('url')** calls for the address of your blog from the **Options** page of the administration panel. The blog title link is for readers that want to come back to the home page after visiting the sub pages.

What if you wanted to show the your blog's description. You'd use **<?php bloginfo('description'); ?>**. But, we're not using it yet.

Add a set of div tags around the blog title area to separate it from the content below it:

** **
<h1><a href="<?php bloginfo('url'); ?>"><?php bloginfo('name'); ?></a></h1>
** **

Again, you used **id=”header”** instead of **class=”header”** because there isn't going to be another header area on your page.

---

# Add Horizontal Menu

Below the header area, type:

This time, we're starting with the XHTML first. As you get more comfortable with creating WordPress themes, you'll notice that you tend to start with the XHTML first.

Add a set of un-ordered list tags to it:

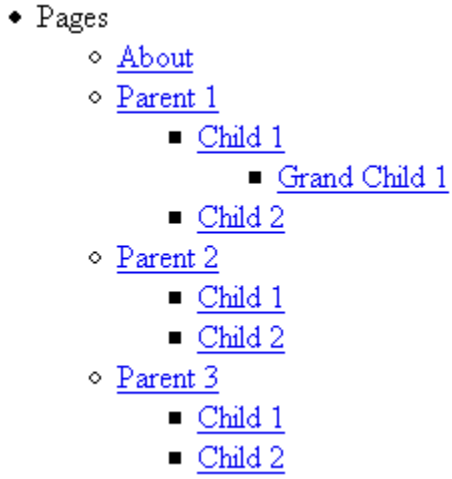
    **<ul>**

    **</ul>**

Un-ordered List = UL. Easy? Now, add <?php wp_list_pages(); ?>

    <ul>
        **<?php wp_list_pages(); ?>**
    </ul>

Save and refresh. What you get is a listed of links with a the word “**Page**” as the title of all those links.



Let's see what that list of links look like behind the scene. Here's something new. On the home page, if you're using Firefox,  right click and select **View Page Source** or you could go to the top of the browser and select **View >  Page Source**. A window with a bunch of codes will pop up.

That's what you've been doing all along. Check out everything that you're generating from the **index.php** file. Here's a partial screenshot of the list of links in code:

```
<div id="menu">
        <ul>
                <li class="pagenav">Pages<ul><li class="pag
<li class="page_item"><a href="http://localhost/test/?page_
        <ul>
        <li class="page_item"><a href="http://localhost/tes
                <ul>
                <li class="page_item"><a href="http://local
                </ul>
</li>
        <li class="page_item"><a href="http://localhost/tes
        </ul>
</li>
<li class="page_item"><a href="http://localhost/test/?page_
        <ul>
        <li class="page_item"><a href="http://localhost/tes
        <li class="page_item"><a href="http://localhost/tes
        </ul>
</li>
<li class="page_item"><a href="http://localhost/test/?page_
        <ul>
        <li class="page_item"><a href="http://localhost/tes
        <li class="page_item"><a href="http://localhost/tes
        </ul>
</li>
</ul></li>      </ul>
</div>
```

Probably, the first thing you notice is the **LI** tags. They are called list-item tags. Those tags nest within **UL** tags. You have an unordered-list (**UL**) and then a set of list-items (**LI**) within it. You can also nest UL within LI too, for example:

```
<ul>
        <li>Parent 1
                <ul>
                        <li>Child</li>
                </ul>
        </li>
</ul>
```

Everything that nest withi UL tags has to be wrapped around by LI tags. Also, notice that I closed the LI and UL tags in order. Here's how you should not close them:

---

```
<ul>
	<li>Parent 1
		<ul>
			<li>Child</li>
		</li>
	</ul>
</ul>
```

Can you see what's wrong with the example above? The closing UL and LI tags are out of order. Now that you know how UL and LI tags work, let's break down our menu.

By default, **wp_list_pages** generate the links list and nests each link or each item within the **LI** tags. It also give the links list a title. That title nests within **h2** tags. Well, we don't want that title because it's going to be a horizontal menu. What we also don't want is the display of child and grandchild links because what we're looking for is one level of links that we can style to display horzontally.

To get rid of the title, child pages, and grandchild pages, first, add **depth=1** to **wp_list_pages**.

```
<div id="menu">
	<ul>
		<?php wp_list_pages('depth=1'); ?>
	</ul>
</div>
```

Notice, depth=1 is wrapped in single quotes, not double quotes. Save and refresh. Now, add **title_li=** to **wp_list_pages**. You'll need to use the **&** sign to add **title_li=** onto **depth=1**.

```
<div id="menu">
	<ul>
		<?php wp_list_pages('depth=1&title_li='); ?>
	</ul>
</div>
```

With the **&** sign, **wp_list_pages** can recognize that you added on two items: **depth=1** and **title_li**, not one long item like **depth=1title_li=**. You didn't give a value to **title_li=** because it has a value by default and that is **<h2>Pages</h2>**. What you did was you got rid of the **h2** tags and "**Pages**" title by not giving it a value.

Save and refresh. You're finished with the horizontal menu. We're moving on to the sidebar. Oooohh, the sidebar. The sidebar is not complicated. Don't be afraid.

# The Sidebar

Before we start the sidebar, here's the condensed version of what you should have so far:

```
<div id="header">
	<h1><a href="<?php bloginfo('url'); ?>"><?php bloginfo('name'); ?></a></h1>
</div>
<div id="menu">
	<ul>
		<?php wp_list_pages('depth=1&title_li='); ?>
	</ul>
</div>
<div id="content">
	<?php if(have_posts()) : ?>
	<div class="navigation"><?php posts_nav_link(); ?></div>
	<?php while(have_posts()) : the_post(); ?>
	<div class="post">
		<h2><a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>"><?php
the_title(); ?></a></h2>
		<div class="entry-date"><?php the_time('F jS, Y'); ?></div>
		<div class="entry-content">
			<?php the_content(); ?>
			<?wp_link_pages(); ?>
			<?php edit_post_link('Edit', '<p>', '</p>'); ?>
		</div>
		<div class="entry-meta">Posted by <?php the_author(); ?> under <?php the_category(',
') ?> with <?php comments_popup_link('No Comments', '1 Comment', '%
Comments'); ?></div>
	</div>
	<?php endwhile; ?>
	<div class="navigation"><?php posts_nav_link(); ?></div>
	<?php endif; ?>
</div>
```

Below the content div, type:

```
<div class=”sidebar”>

</div>
```

Again, we're starting with the XHTML first. class=”sidebar instead of id=”sidebar” because our design has more than one sidebar so we're going to code for more than one sidebar.

Your sidebar is made up of a bunch of list-item (LI) tags that are lost without a set of UL tags so let's add UL to the sidebar:

```
<div class="sidebar">
        <ul>

        </ul>
</div>
```

Looks familiar? You're right. The horizontal menu and the sidebar have the same basic structure. Next, add **wp_list_pages** within the UL tags. Again? Yes, again. But this time, without **depth** and **title_li** because you want all the sub links and the links list title to show this time.

```
<div class="sidebar">
        <ul>
                <?php wp_list_pages(); ?>
        </ul>
</div>
```

Save and refresh. Go to the bottom of the home page. You'll see the same list of links that you were working with earlier for the horizontal menu. That counts as one sidebar block. Each sidebar block can have multiple items, each wrapped in a set of list-item tags.

Let's add another sidebar block, the Catgories block:

```
<div class="sidebar">
        <ul>
                <?php wp_list_pages(); ?>

                <?php wp_list_categories(); ?>

        </ul>
</div>
```

Easy huh? That's because we're using wp_list_categories, which a recent addition to WordPress. For most themes, the categories block look like this:

```
                <li id=”categories”><h2>Categories</h2>
                        <ul>
                                <?php wp_list_cats(); ?>
                        </ul>
                </li>
```

By using **wp_list_categories** instead of **wp_list_cats**, WordPress takes care of the **id=”catgories”** and **<h2>Categories</h2>** title for you. Save and refresh. The next block is the Archives block.

---

For the archives, we have to do some extra work. Here's the Archives block:

```
<div class="sidebar">
	<ul>
		<?php wp_list_pages(); ?>

		<?php wp_list_categories(); ?>

		<li id="archives"><h2>Archives</h2>
			<ul>
				<?php wp_get_archives('type=monthly'); ?>
			</ul>
		</li>
	</ul>
</div>
```

The red highlighted codes are the extra work I'm talking about. The one thing to remember from the red codes is that the sidebar uses **h2** for sidebar titles. It doesn't use h3, h4, h5, or h6. You can make it use h3 or under, but we're not getting there yet. It's not a simply change like switching like this: **<h3>Archives</h3>**. You'll see why later.

Also, notice that **wp_get_archives()** has **type=monthly** within it. If you want to play with it, change it to **type=yearly**.

Save and refresh. You are finished with the first sidebar. Let's start the **second sidebar** under the first one. I went ahead and added the **UL** tags to the secon sidebar **div**.

```
<div class="sidebar">
	<ul>
		<?php wp_list_pages(); ?>

		<?php wp_list_categories(); ?>

		<li id="archives"><h2>Archives</h2>
			<ul>
				<?php wp_get_archives('type=monthly'); ?>
			</ul>
		</li>
	</ul>
</div>

<div class="sidebar">
	<ul>

	</ul>
</div>
```

---

Add four blocks to the second sidebar, the first of the last four blocks is the search block, which default structure looks like this:

```
<li id="search">
	<form method="get" id="searchform" action="<?php bloginfo('home'); ?>">
		<div>
			<input type="text" value="<?php echo wp_specialchars($s, 1); ?
>" name="s" id="s" size="15" />
			<input type="submit" id="searchsubmit" value="Search" />
		</div>
	</form>
</li>
```

**What just happened?**
Here it is in simple words. “I want to start a **list-item** named **search**. After it, I'm starting a **form** tag named **searchform**. After the **form** tag, I'm opening a **nameless div** tag. After the **nameless div** tag, I'll open two inputs. The first **input** tag, named **s**, is where the users will put in the keywords they want to search for. The second **input** tag, named **searchsubmit**, is the actual **Search** button. After the two inputs, I'm going to close the **nameless div** tag. Next, I'll close the **form** tag, named **searchform**, that I started earlier. And finally, I'll close the **list-item**, named **search**, that wraps around everything.

For the second sidebar, you should have:

```
<div class="sidebar">
	<ul>

		<li id="search">
			<form method="get" id="searchform" action="<?php bloginfo('home'); ?>">
				<div>
					<input type="text" value="<?php echo wp_specialchars($s, 1); ?
>" name="s" id="s" size="15" />
					<input type="submit" id="searchsubmit" value="Search" />
				</div>
			</form>
		</li>

	</ul>
</div>
```

Save and refresh. Here's how that looks on your home page:

The second block for the second sidebar:

```
<div class="sidebar">
	<ul>

		<li id="search">
			<form method="get" id="searchform" action="<?php bloginfo('home'); ?>">
				<div>
					<input type="text" value="<?php echo wp_specialchars($s, 1); ?
>" name="s" id="s" size="15" />
					<input type="submit" id="searchsubmit" value="Search" />
				</div>
			</form>
		</li>

		<?php wp_list_bookmarks(); ?>

	</ul>
</div>
```

Save and refresh. That is your list of links. You can add to or delete links from that list on the **Blogroll** page for the WordPress administration panel.

The third block for the second sidebar:

```
		<li id="rss-links"><h2>RSS Feeds</h2>
			<ul>
				<li><a href="<?php bloginfo('rss2_url') ?>">Posts RSS</a></li>
				<li><a href="<?php bloginfo('comments_rss2_url') ?>">Comments
RSS</a></li>
			</ul>
		</li>
```

You started another list-item named **rss-links**. The block title is **RSS Feeds**. With the **rss-links** list-item, you have a set of **UL** tags and two more child list-items. One for the **Posts RSS** link and the other for the **Comments RSS** link. RSS is how readers can subscribe to your blog and read your content without having to visit your blog.

Save and refresh.

The fourth block for the second sidebar:

```
<li id="meta"><h2>Meta</h2>
	<ul>
		<?php wp_register() ?>
		<li><?php wp_loginout() ?></li>
		<?php wp_meta() ?>
	</ul>
</li>
```

This Meta block is the register, log in, and log out block. **wp_register** calls for the **Register** link if the user is not logged in. wp_loginout calls for the **Login** link. After logging in, **wp_loginout** displays a **Logout** link. **wp_meta** doesn't do anything until you actually use it for something. Most of the times, you don't even know it's there.

Note: **wp_register** and **wp_meta** don't need a set of list-item (LI) tags around them, but wp_loginout does.

You're finished with both sidebars. Here's what the codes for both sidebars should have for the sidebars:

```
<div class="sidebar">
	<ul>
		<?php wp_list_pages(); ?>

		<?php wp_list_categories(); ?>

		<li id="archives"><h2>Archives</h2>
			<ul>
				<?php wp_get_archives('type=monthly'); ?>
			</ul>
		</li>
	</ul>
</div>

<div class="sidebar">
	<ul>

		<li id="search">
			<form method="get" id="searchform" action="<?php bloginfo('home'); ?>">
				<div>
					<input type="text" value="<?php echo wp_specialchars($s, 1); ?
>" name="s" id="s" size="15" />
					<input type="submit" id="searchsubmit" value="Search" />
				</div>
			</form>
```

```
		</li>

		<?php wp_list_bookmarks(); ?>

		<li id="rss-links"><h2>RSS Feeds</h2>
			<ul>
				<li><a href="<?php bloginfo('rss2_url') ?>">Posts RSS</a></li>
				<li><a href="<?php bloginfo('comments_rss2_url') ?>">Comments
RSS</a></li>
			</ul>
		</li>

		<li id="meta"><h2>Meta</h2>
			<ul>
				<?php wp_register() ?>
				<li><?php wp_loginout() ?></li>
				<?php wp_meta() ?>
			</ul>
		</li>

	</ul>
</div>
```

# The Footer

Lucky for you, the footer is very simple. Put the footer under the sidebars, here it is:

```
<div id="footer">
	Powered by <a href="http://www.wordpress.org/">WordPress</a>. Theme by <a
href="http://www.wpdesigner.com/">Wpdesigner</a>.
</div>
```

You can put anything in between ** ** and ** ** for your footer message. My message is, “Powered by WordPress. Theme by Wpesigner.”

Save and refresh. After the footer area, add <?php wp_footer(); ?> like this:

```
<div id="footer">
	Powered by <a hr
</div>

<?php wp_footer(); ?>

</body>
</html>
```

Like **wp_meta**, **wp_footer** is not noticeable until you actually use it for something. Using **wp_meta** and **wp_footer** is rare so don't worry about them.

With the footer in place, we're almost finished with the **index.php** file. The last two things we need to add is the About message or area and an id=”wrapper” div that wraps around everything. About area sits in between the horizontal menu and the content. Here's the code for the About area and where it should sit:

```
<div id=”menu”>...</div>

<div id=”about”>
        <p><?php bloginfo('description'); ?></p>
</div>

<div id=”content”>...</div>
```

For this area, we're using something new, the P tags. P means paragraph. That's pretty much it.

I named that div **“about”** rather than “**description**” because you might want to put a lot of stuff there, more than just a simple blog description. You can change the message of **bloginfo('description')** at the **Options** page of the administration panel. If you want your description to stay the same, but also want to display a different message in the **About** area, simply not use **bloginfo('description')** for the **About** message.

For the wrapper div, add ** ** after **<body>** and the closing ** ** before **</body>**.

Save and refresh. That's it. According to the design we're working with, you are finished with everything in the index.php file. Let's move on to styling, using **style.css**.

# Style.css

Before we start messing with this file, let's review what we have in it so far. By the way, you can save and close the **index.php** Notepad now. Here's what we have for style.css so far:

```
/*
Theme Name: Wrath
Theme URI: http://www.wpdesigner.com/
Description: Light version of Wrath, used for WordPress tutorial.
Version: 1.0
Author: Small Potato
Author URI: http://www.wpdesigner.com/
*/
```

At this point, you need to use Firefox and Internet Explorer. Open up both browsers and go to your blog's home page.

---

Below what you have in style.css right now, type:

```
body, h1, h2, h3, h4, h5, h6, address, blockquote, dd, dl, hr, p, form{
	margin: 0;
	padding: 0;
}
```

The codes above are CSS codes used in this format:

```
selector {  property: value; }
```

But as you can see from your first block of CSS codes, you can have multiple selectors working with multiple properties. You can also have multiple values for each property, but we're not getting there yet.

You could code it this way:

```
body{
	margin: 0;
	padding: 0;
}

h1{
	margin: 0;
	padding: 0;
}
```

and so on... for the rest of the selectors.

But, why do that when you can combine all the selectors that share the same properties and values? That's what's going on with the first block of CSS codes I gave you.

Just like working with the **index.php** file, save and refresh. The spacing in between each item on your home page is gone. Don't worry, you're simply resetting things so you can have all the spacings look the way you want them to.

To explain **margin** and **padding**, let's go back to the **div** tags or the invisible boxes. Margin is spacing outside of the box. Paddings is spacing inside of the box.

Now, we're going to reset the font size, font family, and several other stuff for the body.

```
body{
        font-family: verdana, arial, helvetica, sans-serif;
        font-size: 12px;
        text-align: center;
        vertical-align: top;
        background: #ededed;
        color: #000;
}
```

**font-family: verdana, arial, helvetica, sans-serif;**
It basically says, “Use the Verdana font as the default font for every word on my blog. If the user's computer doesn't have Verdana, use Arial. If the user's computer doesn't have Arial, use Helvetica. If there's no Helvetica, use Sans-serif.” The semi-colon at the end of the line is how you end a line of code in CSS. All the codes sit within { and }. It's not that complicated, picture using < and > like you did for XHTML.

**font-size: 12px;**
The font size for every word on my blog will be 12 pixels. It's for every word because I'm using the body selector. If I were to use the h1 selector, only the blog's title or the main heading of the page will have a 12 pixel font size

**text-align: center;**
Center everything. Usually, you don't want to do that, you want everythint to start from the left, but this is for layout purposes and we'll fix it later on.

**vertical-align: top;**
Start everything from the top down, not the middle down, and definitely not from the bottom.

**background: #ededed;**
My background color is #ededed. #ededed is a hex code for light gray.

Furthermore:

- #FFFFFF or #ffffff is a hex code meaning white. While coding, all colors have a specific hex code.
- Hex codes range from #ffffff, #eeeeee, #dddddd, #cccccc, #bbbbbb, #aaaaaa, #999999, and #888888, to #777777, #666666, #555555, #444444, #333333, #222222, #111111, and finally #000000 (black).
- The first two digits represent Red. The third and fourth digits represent Green. The fifth and sixth digits (that's right) Blue. You're getting the hang of this aren't you?
- Hence, #ff0000 is Red. #00ff00 is Green. #0000ff is Blue.What, no yellow? Calm down. #ffff00 is Yellow, a combination of Red and Green.

---

**color: #000;**
All text / words will be black. How come the hex code is only three digits this time? 3-digit is a short cut for six-digit hex codes that are exactly the same. #000 means #000000. #f7c means #ff77cc. In three digits, each digit represents two identical numbers. What about #ff77cb? You can't use a short cut for that because #f7cb is not a short cut.

Save and refresh**.** Next, style the **table** tag, add:

```
table{
	font-family: verdana, arial, helvetica, sans-serif;
	font-size: 12px;
}
```

The table tag is rarely used, but we're styling it just in case you want to add the WordPress calendar to the sidebar, which does use the table tag. For the **table{}** selector, the **font-family** and **font-size** values are identical to that of the **body{}** selector. However, we didn't group them together like **body, table{}** because we don't want to use the **text-align**, **background**, and **color** for the **table{}** selector.

Save. If you refresh the page, you won't see any changes because you have not used **table** in **index.php**. Next, style the main and sub headings:

```
h1, h2, h3, h4, h5, h6{
	font-family: georgia, arial, helvetica, sans-serif;
	font-size: 16px;
	font-weight: bold;
}
```

Save and refresh to see the heading changes. Next, we style the links:

```
a{
	text-decoration: underline;
	color: #8f3939;
}

a:hover{ text-decoration: none; }
```

**a{}**is for the link or achor tag. **text-decoration: underline;** makes sure each link will be underlined. With **a:hover{}**, you are styling everything that happens when you put the mouse over a link.

So, **a:hover{ text-decoration: none; }** means don't underline the link when you place your the mouse / cursor over it. Let's take it a step further with link styling and style for image links. By default, image links have a border around each of them. To get rid of the ugly default border applied to image links, use:

```
a img{border: 0;}
```

---

Now, we're going to style the paragraph (**P**) tags. Although we used it only once while coding, we've been generating a lot of P tags. Go to **View > Page Source** to see all the **P** tags. Since we stripped all the spacing from the P tags. Let's give it some padding:

```
p{ padding: 10px 0 5px; }
```

Save and refresh to see the change. There's a lot more little stuff here and there that you need to style for before you get to the layout part of the design. So, type everything you see below without further explanations from me:

```
blockquote{
	margin: 10px 0 0;
	border-top: 2px solid #ddd;
	background: #f5f5f5;
}

blockquote p{ padding: 10px; }

blockquote blockquote{
	float: none;
	width: auto;
	margin: 0 10px;
	background: #fff;
}

dd{
	padding: 0 0 0 20px;
}

form, input, textarea{
	font-family: verdana, arial, helvetica, sans-serif;
	font-size: 12px;
}

p img{
	max-width: 100%;
}

img.centered{
	display: block;
	margin-left: auto;
	margin-right: auto;
}
```

```
img.alignright{
	margin: 3px 0 2px 10px;
	padding: 4px;
	border: 1px solid #ededed;
	display: inline;
}

img.alignleft{
	margin: 3px 10px 2px 0;
	padding: 4px;
	border: 1px solid #ededed;
	display: inline;
}

.alignleft{float: left;}

.alignright{float: right;}

.clear{margin: 0; padding: 0; clear: both;}

small{
	font-size: 11px;
}
```

Save and refresh to see the changes. You can go to w3schools.com for more CSS information. Now, we'll move on to the layout part.

Remember the last **div** you added to the **index.php** file? We'll use that to help us position the entire layout. Here's how:

```
#wrapper{
	width: 980px;
	margin: 0 auto;
	text-align: left;
}
```

We're styling the **id=”wrapper”** div. How would you style a class then? Use a period. id=”wrapper” is **#wrapper** in CSS and class=”wrapper” is **.wrapper** in CSS.

Save and refresh to see the change. First, we gave **#wrapper** a width, 980 pixels. Second, **margin: 0 auto;** means zero top margin, auto right margin, zero bottom margin, and left auto margin. **0 auto** is the short way to write **margin: 0 auto 0 auto;**. Treat auto as a number like 0. You'll see that the margins are styled clockwise, 0 for top, auto for right, 0 for bottom, and auto again for left.

Auto left and autor right margins are for centering the 980 px wrapper div in Firefox. Centering the

---

wrapper div in Internet Explorer has been taken care of earlier, by using **text-align: center;** within **body{}**.

Third, we reset the text-align to left, one more time in **#wrapper{}** because text-align: center; in **body{}** was centering everything and we don't want everything to be centered.

After **#wrapper{}**, add:

```
#header{
	float: left;
	width: 980px;
	border-top: 5px solid #000;
	background: #fff;
}
```

Thanks to **float: left**, the entire header area is floating left, but you can't tell that it is floating left because of the 980 px width that takes up the entire width of the layout.

Just like margin, **border-top: 5px solid #000;** is another example of using multiple values for one property. The first value determines how thick the border is going to be; the second value is the style of the border (border style can be **solid**, **dotted**, **dashed**, etc.); the third value is the color of the border. Again, **#000** means **#000000**, which means we want the border to be black. **Background: #fff;** means white background. Save and refresh.

Next, we style the blog title, which is sitting within the **H1** tags:

```
#header h1{
	padding: 10px;
	font-size: 18px;
	font-weight: normal;
}
```

**#head h1{}** is not the same as **#header, h1{}**. When you use a comma to separate two selectors, you are grouping them. When you don't use a comma to separate them, you are trying to be specific. **#head**, **h1{}** means **#header** and **h1** will have the same styles. On the other hand, **#header h1{}** means only the **h1** tag of the **#header div** will have this style.

**padding: 10px;** - This adds exactly 10 pixels around whatever object you're trying to style. **padding: 10px** actually means **padding: 10px 10px 10px 10px**, which means 10 pixel padding around all four sides. It's another CSS shorthand. If all the shortcuts or shorthand coding are confusing you, type everything out the long way.

**font-size: 18px;** resets the font size for h1 within the #header div or invisible box.

**font-weight: normal;** resets the font weight. Font weigth can be bold or normal.

---

Save and refresh to see the change. Don't worry about everything else. Some parts of the home page might look odd or broken. That's because you don't have the entired page styled right now.

Next, we style the menu:

```
#menu{
	float: left;
	width: 980px;
	border-top: 1px solid #ededed;
	background: #fff;
}
```

Just like the **#header** div, you're making the **#menu** float left with a 980 pixel width. Unlike **#header**, #menu's top border is only 1 pixel thick. **background: #fff;** means white background for the #menu div. #fff = #ffffff.

**Tip**: Never use left and right border / padding for floating div(s) with a definite width. Let's take the **#header** div for example. Why didn't I use **padding: 10px;** within **#header{}** instead of within **#header h1{}**? It would make more easier and it would make more sense for me to do something like this:

```
#header{
	float: left;
	width: 980px;
	padding: 10px;
	border-top: 5px solid #000;
	background: #fff;
}
```

But, I didn't because not all browsers interpret your codes the same way. One browser might count the 10 pixel padding as a part of the 980px width. Another browser might count it as additional spacing. Additional spacing would mean 980px + 10px left padding + 10px right padding = 1000px total.

That is exactly why I wanted you to work with **Firefox** and **Internet Explorer**. You can't simply code or style for one browser because not all of your blog readers use the same tool to view your site. While styling, if there's any inconsistency between the two browsers, work around it. Using **padding: 10px** for **#header h1{}** instead of **#header** is a work-around.

(I'm aware that Firefox and IE are the only two browsers in the world. While it's important to make your site work or display properly across multiple browsers, it's also important not to waste your time squashing every bug across ten to twenty browsers.) Moving on...

Under **#menu{}**, type:

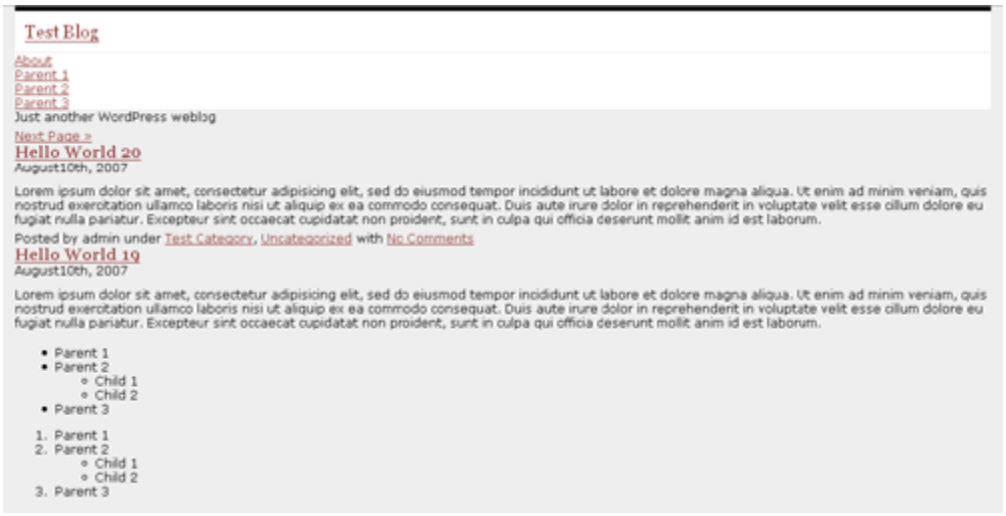
```
#menu ul{
        list-style:none;
        margin:0;
        padding:0;
}
```

**list-style: none;** gets rid of the bulleted style for the menu links that we want to display as a horizontal menu.

**margin: 0;** and **padding: 0;** strips all the default spacing for the unordered-list (UL) tag. **Tip**: Don't reset the margin and padding for UL as a part of:

```
body, h1, h2, h3, h4, h5, h6, address, blockquote, dd, dl, hr, p, form{
        margin: 0;
        padding: 0;
}
```

Save and refresh. So far, your theme should look similar to the screenshot below:




Next, we make the list-items within UL line up horizontally:

```
#menu ul li{
        float: left;
}
```

#menu ul li means list-item tags of the unordered-list within the #menu div. Save and refresh to see the change.

On to styling each link within each item to pretty up our horizontal menu:

```
#menu ul li a{
	display: block;
	padding: 10px;
	text-decoration: none;
	color: #777;
}
```

**#menu ul li a** means we're targetting link tags or **A** tags within **list-items** of the **unordered-list** in the **#menu** div.

Think of **display: block;** as telling links to act like **div**s or invisible boxes. Now that the links are acting like **div**s, we can style and have them display properly. Without **display: block;** the styles for links within a horizontal menu won't always come out right.

**padding: 10px;** - You already know what this is. But again, I'm using it for **#menu ul li a{}** instead of **#menu ul li{}** because the browsers will not properly interpret the padding style for floating list-items.

**text-decoration: none;** strips the default underline for all links, from the horizontal menu links.

**color: #777;** simpy resets the link color for the links within the horizontal menu only. The rest of the links on your page are not affected by this.

Save and refresh to see padding added to your horizontal menu.

Because we stripped the underline from the links, the horizontal links look like a simple line of text. At this point, it's hard to tell they really are links. So, let's go beyond the original design. Add something more to the link styles to help people realize that they are links. We're going to add a border to the right of each link like this:

```
#menu ul li a{
	display: block;
	padding: 10px;
	border-right: 1px solid #ededed;
	text-decoration: none;
	color: #777;
}
```

Save and refresh. That is a perfect example of simple modifications that help the design look and work much better than the original idea. While we're still messing with the menu, let's add some more toys to this thing:

```
#menu ul li a:hover{
	background: #f9f9f9;
}
```

---

Save, refresh, and move your mouse over one of those links to see the background color change. **a:hover** allows you to tell the links how to look like while a user hover the cursor over it. That's one of the little things that you can do to let users interact with your blog's design or layout. But remember, don't go overboard with all the little affects.

Now, we move on to the **about** div. But before we style it, grab the **about.gif** image from the first **Part 2 – How to Slice a WordPress Theme** tutorial. Create a new folder within your "**wrath**" folder. Name it "**images**" and put the **about.gif** image in there. If you didn't follow along with **Part 2**, don't worry; I included the **about.gif** in the images folder of your tutorial folder. Moving on to the styling...

Type:

```
#about{
	float: left;
	width: 980px;
	border-top: 1px solid #5f0000;
	border-bottom: 1px solid #5f0000;
	padding: 0 0 14px 0;
	font-family: arial, helvetica, sans-serif;
	font-size: 18px;
	line-height: 30px;
	background: #c00 url(images/about.gif) repeat-x left bottom;
	color: #fff;
}
```

(From this point on, I'll focus mainly on explaining the codes that you haven't come across yet.) **background: #c00 url(images/about.gif) repeat-x left bottom;** means background color will be **#c00**, which is actually **#cc0000**. On top of the background color, I want my background image to be **about.gif**, which is in the "**images**" folder. Also, I want my background to start from the left bottom corner of the **#about** div and repeat itself horizontally (repeat-x). X is across and Y is up and down.

**padding: 0 0 14px 0;** means 14 pixel bottom padding.

We've also changed the line-height to 30 px because the font size for #about div is 18 px. At 18 pixels, the words are pretty big. We want a big line-height to go along with it so it'll be easier to read. If you have a long about message that takes up two or more lines, you will see the change in line-height.

Save and refresh.

Let's get to the **P** tag within the **#about** div:

```
#about p{
	padding: 10px 10px 5px 10px;
}
```

In order: top = 10px, right = 10px, bottom = 5px, left = 10px. Save and refresh. **About** div is complete.

---

Before we move on to the next div to style. Open **index.php**, add a set of div tags around the content and sidebar like this:

```
<div id="container">
        <div id="content">... </div>
        <div class="sidebar">...</div>
</div>
```

We need this extra container div because we'll be using a background image under the content and sidebar areas. Save the **index.php** file and close it afterward.

Back to the **style.css** file. Type:

```
#container{
        float: left;
        width: 980px;
        border-top: 5px solid #000;
        background: #fff url(images/bg_container.gif) repeat-y 488px 0px;
}
```

The **bg_container.gif** image is also in the "**images**" folder. Make sure that you move or copy and paste that image into your own "**images**" folder. This time, we are positioning the background image using exact numbers, instead of **left** and **bottom**. 488 px and 0 px, the first number is how many pixels to place the background image from the left edge of the container div. The second number is how many pixels to place the image from the top edge. Repeat-y is tell the background image to repeat itself vertically. Tip: If you want a certain background image to repeat horizontally and vertically, simply not use repeat-y or repeat-x at all. It repeats itself in all directions by default.

Save and refresh to see your background image. Once you have the content and sidebars in the right places, it'll look like your page is completely divided into three columns.

Moving on to content and sidebars...

```
#content{
        float: left;
        width: 488px;
}

.sidebar{
        float: left;
        width: 245px;
        margin: 0 0 0 1px;
}
```

For the **sidebar div**s, we're using the period instead of the pound sign because the **sidebar div**s in

**index.php** are classes, not IDs. With the dotted background image in mind, we have to give each sidebar a 1 pixel margin so they don't overlap the visible part of the background image even though the sidebars simply lying on top of the background image.

Now, we're going to style everything within the **content** div, starting with the **navigation** div. (Remember, we used the navigation div in two places within the **index.php** file.)

Place the following between **#content{...}** and **.sidebar{...}**:

```
.navigation{
	margin: 10px 10px 0;
	border-top: 1px solid #e0dcb8;
	padding: 5px 10px 6px;
	background: #fdfbe7 url(images/bg_navigation.gif) repeat-x left bottom;
	line-height: 24px;
}
```

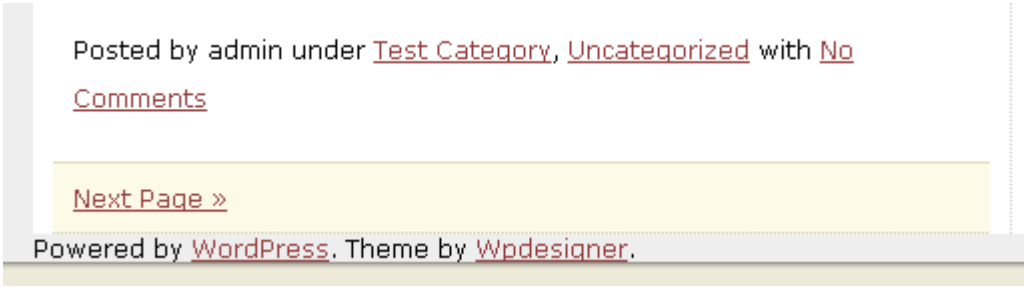
Save  and refresh. Now, we're going speed up. Add all the styles listed below after .**navigation{}**. I'm not going to explain them. It's best if you add them one by one, save and refresh to see the change. You'll get to see what each code does for yourself.

```
.post{
	padding: 10px 20px;
}

.post h2{
	font-size: 24px;
	font-weight: normal;
}

.post h2 a{
	color: #000;
}

.entry-date{
	padding: 10px 10px 0 10px;
	color: #666;
}

.entry-content{
	line-height: 24px;
}

.entry-content h2, .entry-content h3, .entry-content-h4, .entry-content h5{
	padding: 10px 0 5px;
}
```

---

```
.entry-content h2 a{
	color: #8f3939;
}

.entry-content h3{
	font-size: 18px;
	font-weight: normal;
}

.entry-content h5{
	font-size: 14px;
}

.entry-content h6{
	font-size: 12px;
}

.entry-meta{
	padding: 10px 0 0 0;
	line-height: 24px;
}
```

Your theme should look a little bit more organized now. But there's one problem and here it is:




The bottom area of the content div is cluttered. Theres no spacing for the navigation div. Luckily, it's not a big problem and I actually planned it by leaving **padding: 0 0 10px 0;** out of **#content{}**. Go back to **#content{}** and add the padding. Save and refresh.

Not everything will go according to plan. It's also true with coding. That's why it's important to test your design while you code and / or style it. You can't code everything without testing, save, hit one refresh, and expect a perfect layout.

Moving on to the sidebar...

```
.sidebar ul{
	list-style: none;
	margin: 0;
	padding: 0;
}

.sidebar ul{
	margin: 0 0 10px;
}

.sidebar ul li{
	padding: 10px 10px 0;
}

.sidebar ul li h2{
	border-top: 1px solid #dfdfdf;
	padding: 8px 10px 9px;
	font-family: Georgia, Arial, Helvetica;
	font-size: 16px;
	font-weight: bold;
	background: #f8f8f8 url(images/bg_sidebar_h2.gif) repeat-x left bottom;
}
```

You'll see that even I made a mistake. Open up **index.php**. Add **'title_li=<h2>Pages</h2>'** to **wp_list_pages()** of the first sidebar like this:

```
<?php wp_list_pages('title_li=<h2>Pages</h2>'); ?>
```

Now, continue with the styles in **style.css**.

```
.sidebar ul ul{
	margin: 0;
	padding: 6px 10px 0;
	line-height: 24px;
}

.sidebar ul ul li{
	padding: 0;
}

.sidebar ul ul ul{
	padding: 0 0 0 10px;
}
```

---

Now, your sidebars should look very organized too. Notice, I styled three levels of UL and LI tags. Level 2 resets Level 1 rules. Level 3 resets Level 2 rules.

Let's get to the footer shall we?

```
#footer{
	float: left;
	width: 980px;
	padding: 10px 0 10px 0;
	border-top: 5px solid #000;
	text-align:center;
	line-height: 24px;
	background: #ededed;
}

#footer a{
	text-decoration: none;
}
```

Save and refresh. You will need to create and work on several smaller files before you can finish off the **style.css** file.

# Creating the sub template files

The following files are based on **index.php**. What you'll do first is split up **index.php** into four files: **header.php**, **index.php**, **sidebar.php**, and **footer.php**.

- Create a new text document, rename it to **header.php**
- Open up **index.php**, copy everything from the top to , paste it in header.php, and save the header file. Make sure the header files include 
- Go back to index and replace the whole area that you just copied and pasted, with **<?php get_header(); ?>**. Save index and refresh the page to see if your theme is still working.
- Create a new text document, rename it to **sidebar.php**
- Copy everything of the first and second sidebar to the sidebar file
- In index, replace that whole area with **<?php get_sidebar(); ?>**. Save and refresh to make sure that your theme is working without errors.
- Create a new text document, rename it to **footer.php**
- Copy everything from  (of container) to </html>, paste it in the footer file
- In index, replace that whole area with **<?php get_footer(); ?>**. Save the Notepad and refresh the page to see your theme working without errors.

At this point, you should have the following codes in index:

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

<div id="content">

	<?php if(have_posts()) : ?>

	<div class="navigation"><?php posts_nav_link(); ?></div>

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

	<div class="post">
		<h2><a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>"><?php
the_title(); ?></a></h2>
		<div class="entry-date"><?php the_time('F jS, Y'); ?></div>

		<div class="entry-content">

			<?php the_content(); ?>
			<?wp_link_pages(); ?>
			<?php edit_post_link('Edit', '<p>', '</p>'); ?>

		</div>
		<div class="entry-meta">Posted by <?php the_author(); ?> under <?php the_category(',
') ?> with <?php comments_popup_link('No Comments', '1 Comment', '%
Comments'); ?></div>
	</div>

	<?php endwhile; ?>

	<div class="navigation"><?php posts_nav_link(); ?></div>

	<?php endif; ?>

</div>

<?php get_sidebar(); ?>

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

## More sub template files...

- Create a new file, name it **archive.php**
- Copy everything in the index file to archive.php
- Within the archive file, change **the_content** to **the_excerpt**. Now, your archive pages will display excerpts only. If you don't want that then don't do this step. If you don't use an archive.php file to customize your archive pages, WordPress will rely on the index.pnp template file to organize the archive pages. So, the archive.php file is only necessary if you want to customize the archive pages.
- Create a new file, name it **search.php**
- Copy everything in the index file to search.php. Change the_content to the_excerpt. Yes, that's exactly the same as archive.php. But now, you're customizing the search result pages. I'm simply showing  you what's possible with only one small change. You can have a completely different layout for the search result pages, if you want to that is.
- Create a **404.php** file. It's for when users get lost and go to a page that doesn't exist of has been deleted.

Your **404** file should contain the following codes:

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

<div id="content">

	<div class="post">
		<h2>Not Found</h2>
		<div class="entry-content">
			<p>The page you are looking for is not here.</p>
		</div>
	</div>

</div>

<?php get_sidebar(); ?>

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

The structure is almost exactly like a blog post, but the message is static. You'r not using The Loop with it. You can customize the **404** message, “The page you are looking for is not here,” to make it say what you want it to say.

Next, create **page.php** and **single.php**. Work on **single.php** first. The **single.php** template file controls the look of single post view. It's also the one that controls where the comments get displayed.

- Copy everything from the **index** file to the **single** file.
- Remove the word "**with**" and the code to call for the comments number from the single.php file. The comments number will show up in the comments template later on.
- Replace <?php posts_nav_link(); ?> with:

<?php next_post_link('« %link') ?> <?php previous_post_link('%link »') ?>

Remember, you have to do this twice, once for the top and again for the bottom **navigation div**.

- Add the comments div and comments template function to single.php like this:

```
<div id="comments">
	<?php comments_template(); ?>
</div>
```

It's important that you use "**comments**" as the ID this time around. And since you don't have a **comments.php** file yet, the **comments_template** function will call for the default comments template from the default WordPress theme. That one will do us no good. Use the comments.php file that I've included with this tutorial. Check your tutorial folder. Copy that file to your theme's folder.

I'm not going to explain the stuff in the comments.php file because you need more experience before you can start messing with it.

Now, let's style it:

```
#comments{
	margin: 10px;
	padding: 10px;
	border: 1px solid #f5f5f5;
}

#comments ol{
	list-style: none;
	margin: 10px 0;
	padding: 0;
}

#comments ol li{
	list-style: none;
	margin: 10px 0 0;
	padding: 0 0 10px;
	border-bottom: 1px solid #ededed;
	line-height: 24px;
}
```

```
#comments span.comment-author{
	font-weight: bold;
}

#respond{
	padding: 10px;
	background: #f9f9f9;
}
```

The **respond div** is in the **comments.php** file. The comments template is always tricky and it has several hidden parts that you have to test. It's best to post several test comments include a comment on password-protected posts. What you can also do is go to the Options page of the administration panel and check “Users must be registered and logged in to comment .“ That will force the comments template to display this message:

**Your must be logged in to post a comment.**

Remember,  you're just requiring people to login in order to test that the template styling for that message. Afterward, you can go back and uncheck, “Users must register...” The comments template will give you a different message for password-protected posts. Those hidden messages and templates within the **comments.php** file are all customizable, but they are connected with a lot of important PHP codes so you'll need more experience before you can mess with it.

# The page.php file

Pages like About, Parent 1, and etcetera are also posts, but they are organized separately. Page.php file controls the template of pages like About, but it's not necessary to have a page.php template file.

- Copy everything from the **index.php** file to the **page.php** file.
- Remove the both navigation divs, entirely, which includes the posts_nav_link function.
- Remove the entire entry-date area.
- Remove the entire entry-meta area.

That's it, you're finished with the **page.php** file and finished with this tutorial. All you need to do now is validate your pages using the XHTML and CSS validators you bookmarked at the beginning of this tutorial. If you've installed the Firefox Web Developer add-on, that would make the validation process much easier.

**Suggested readings**:
How to Validate
How to Create a Theme Screenshot
Widgetizing sidebar
WordPress Theme Checklist

If you need help, find me at **http://www.wpdesigner.com/forums**.

---