DIGGING INTO
WordPress
by Chris Coyier & Jeff Starr
How to set things up right
Building themes and how they work
Keeping sites secure and optimized
Making the most of WordPress

Acknowledgements

- Thank you to Thane Champie and James Starr for their help with proofreading.
- Thanks also to the many readers who have helped with further improvements.
- Thank you to everyone who allowed screenshots to be printed in this book.
- Thank you to the incredible WordPress community for making WordPress #1.
- And most of all, thanks to you, the reader, for sharing this adventure with us.

Chris would like to thank

- My mom for all the excellent printing advice (and for being a good mom).
- Jeff Penman for insisting the book was a good idea.
- Tim Chatman for giving me the time and assistance I needed to get it finished.

Jeff would like to thank

- My wife, Jennifer, for her loving support and encouragement.
- My two children, Josh and Lorelei, for being so awesome.
- Friends, family, and everyone who helps along the way.

Disclaimer: links and references to external or third-party websites and resources are provided solely for the reader's convenience. Following links to other sites is done at your own risk and the book's authors, publishers, and all related parties accept no liability for any linked sites, resources, or related content.

DIGGING INTO WORDPRESS
By Chris Coyier and Jeff Starr
Version 5.8 • August 2021

© 2009–2021 Perishable Press Books

Visit us online @ DigWP.com

DIGGING INTO WORDPRESS

CHRIS COYIER & JEFF STARR

Short URLs

The URLs in this book are so called "short URLs." They look like this: https://digwp.com/u/1 – When you click on one (PDF people) or enter one into a browser (Book people), you will be instantly redirected to the URL we are trying to get you to. What's up with that? Are we trying to drive traffic to our own site? Nope, we are trying to do two things:

1) **Make it easier for you** (you don't need to type in long awkward URLs)

2) **Make it easy for us** (it is easier to typeset and design around short URLs than long ones)

5.8

That is the current version of WordPress at the time this book was published. So if we say something like "...the current version of WordPress," we are talking about 5.8. If we need to mention an older version, we'll be specific about that.

So what if you are reading this and 5.9 is already out? **Don't worry about it!** The information in here will still be valid. WordPress does a good job about not breaking existing stuff for new versions.

But there will be changes, and we intend to keep this book updated with those new things. All current owners of this book will get free PDF updates as it is updated!

Subscribe to the DigWP Newsletter

Check out DigWP on Facebook

See that? That's dog food. It's a metaphor.

We, the authors of Digging Into WordPress, eat our own dog food. We aren't just here to stand on a pedestal and preach about how you should do things. We practice these things in the sites we work on every day.

Much of what you will read in this book is put into practice on the WordPress blog that accompanies this book.

https://digwp.com/

Contents

1 Welcome to WordPress

1.1.1 Welcome 9

1.1.2 Why WordPress is Amazing 9
1.1.3. How to Set up and Configure WordPress 10
1.1.4 How to Implement Advanced Functionality 10
1.1.5 How to Optimize and Secure WordPress 11
1.1.6 How to Maintain Your WordPress Site 12
1.1.7 Don't Worry 12

1.2.1 So, You've Never Heard of WordPress 12

1.2.2 One Template, Many Pages 14
1.2.3 Powerful, Flexible and Extensible 14

1.3.1 Key Components of a WordPress Site 15

1.3.2 WordPress Core Files 15
1.3.3 The WordPress Database 15
1.3.4 The Back End 17
1.3.5 The Front End 17

1.4.1 Tools of the Trade 17
1.4.2 A Domain Name 17
1.4.3 Web Host / Server 18
1.4.4 Text / Code Editor 19
1.4.5 FTP Program 19

2 Setting Up WordPress

2.1.1 The Famous Five Minute Install 23
2.1.2 Where To Install? 23
2.1.3 Checking Default Performance and Proper Functionality 24
2.2.1 OK, I'm In. Now What? 25
2.2.2 Just Publish Something! 25
2.2.3 Go Look At It! 25
2.2.4 The Plan 26
2.3.1 Permalinks: Your URL Structure 26
2.3.2 HTAccess 27
2.3.3 Which Style of Permalinks? 28
2.3.4 Pick One and Stick With It 29
2.3.5 SEO Consideration: Mind Your Post "Slugs" 29
2.4.1 Categories and Tags 30
2.4.2 They are Basically the Same 32
2.4.3 Use Only One Category Per Post 32

2.4.4 Use Multiple Tags Per Post 33
2.4.5 Don't Go Overboard! 33
2.4.6 You Don't Need to Use Them At All 34
2.4.7 Custom Taxonomies 34

2.5.1 Users and Administrators 37

2.5.2 Add a New Account for Yourself 39

2.6.1 Choosing the Perfect Theme 40

2.6.2 Where to Find Awesome Themes 40
2.6.3 Previewing Themes 41
2.6.4 Key Things to Look For in a Theme 41

2.7.1 Getting Started with Plugins 44

2.7.2 Installing and Activating Plugins 44
2.7.3 Difference Between Disabling and Uninstalling 45
2.7.4 Recommended Plugins 46

3 Anatomy of a Theme

3.1.1 Understanding Theme Files 51

3.1.2 Every Theme is Different 51
3.1.3 Commonly Used Theme Files 51
3.1.4 How Theme Files Work Together 54

3.2.1 Understanding Different Page Views 54
3.2.2 Page Views are for Pages 55
3.2.3 Single Views are for Posts 55
3.2.4 The Many Faces of Archive Views 56
3.2.5 How WordPress Decides Which File to Use for Rendering the View 56
3.3.1 Kicking It Off with the Header 58
3.3.2 The DOCTYPE and HTML Attributes 58
3.3.3 META Elements 59
3.3.4 The Title 59
3.3.5 Link Elements 61
3.3.6 The wp_head() Function 64
3.3.7 Template Tags 64
3.4.1 The WordPress Loop 67
3.4.2 The Loop in Plain English 68
3.4.3 The Loop Just Knows... 69
3.4.4 Some Common "Loop Only" Functions 70
3.4.5 Some Common "Outside Loop" Functions 71
3.5.1 Comments 71
3.5.2 The comments.php File 71
3.5.3 Selective Inclusion for Different Views 72

3.6.1 The Sidebar 74
3.6.2 Purpose and Placement 74
3.6.3 Popular Sidebar Functions 75
3.6.4 Widgets, Widgets, Widgets 78
3.7.1 The Search Form 79
3.7.2 Why is This a Separate File? 79
3.7.3 Alternatives to WordPress Search 79
3.8.1 The Footer 81
3.8.2 The wp_footer() Hook 81
3.8.3 Mini Footers / Mega Footers 83
3.9.1 Theme Functions 83
3.9.2 Functions are for Specific Themes 83
3.9.3 Advantage Over Core Hacks 84

4 Theme Design & Development

4.1.1 Customizing the Loop 87
4.1.2 Customizing the Loop with query_posts 88
4.1.3 Customizing the Loop with WP_Query 90
4.1.4 Customizing the Loop with get_posts 92
4.1.5 The Loop Doesn't Care About Markup 93
4.1.6 The Power of WP_Query 94
4.1.7 Displaying Different Numbers of Posts 95

4.1.8 Excluding Specific Categories 96
4.1.9 Changing the Sort Order 96
4.1.10 Show Specific Pages, Embed a Page within a Page 97
4.1.11 Using Multiple Loops: An Example 97

4.2.1 Sidebars and Footers 100

4.3.1 Menus, Archive Lists & Tag Clouds 103

4.3.2 Page-Specific Menu Styles 105
4.3.3 Create the Perfect Archives Page 107
4.3.4 Impress Your Visitors with a Tag Cloud 108

4.4.1 Side Content & Useful Menu Items 109

4.4.2 Displaying Recent Comments 109
4.4.3 Displaying Recent Posts 111
4.4.4 Listing Popular Posts 112
4.4.5 Listing Recently Modified Posts 112
4.4.6 Listing Random Posts 113
4.4.7 Import and Display Twitter 113
4.4.8 Import and Display Delicious 116
4.4.9 Import & Display Other Feed Content 117

4.5.1 Creating and Using Child Themes 119

4.6.1 Styling Your Theme 120

4.6.2 Different Inclusion Methods 121
4.6.3 To Reset or Not To Reset? 122

4.7.1 Using Multiple Themes 125

4.8.1 Widgetizing 127

5 Extending Functionality

5.1.1 Extensibility 131

5.1.2 Extending WordPress with Plugins 131
5.1.3 A Plugin for (Almost) Everything 131
5.1.4 Do You Need a Plugin? 133
5.1.5 Choosing the Perfect Plugin 135

5.2.1 Plugin Usage and Maintenance 136

5.2.2 Sequential Installation 137
5.2.3 Keep Plugins Up-To-Date 137
5.2.4 Subscribe to Plugin Comment Threads 138
5.2.5 Getting Help with Plugins 138
5.2.6 Diagnosing Plugin Conflicts 139
5.2.7 Disabling and Uninstalling Plugins 139

5.3.1 Extending WordPress with Custom Functions 142

5.3.2 Plugins vs. Theme Functions (functions.php) 143
5.3.3 Examples of Useful Theme Functions 143
5.3.4 Example #1: Easy Admin Buttons for Comments 144
5.3.5 Example #2: Sitewide Shortcode Functionality 145
5.3.6 Example #3: Moving Plugins to functions.php 146

5.3.7 Example #4: Creating Plugins from Functions 146

5.4.1 Other Ways to Extend & Customize 147

5.4.2 Functions Within Theme Files 149
5.4.3 Hacking the WordPress Core 150

5.5.1 WordPress as a CMS 151

5.5.2 Working With Custom Fields 152
5.5.3 Users, Roles and Permissions 155
5.5.4 Categorizing, Tagging, and Custom Taxonomies 156
5.5.5 Page Templates 158
5.5.6 Page, Category, and Tag Hierarchies 159
5.5.7 Dynamic Menus 160

5.6.1 Extending CMS Functionality 161

But wait, there's more... 161

6 Working with RSS Feeds

6.1.1 Working with RSS Feeds 163

6.1.2 The Pros and Cons of Delivering Feeds 163

6.2.1 Different Types of WordPress Feeds 164

6.2.2 Posts Feed 164
6.2.3 Comments Feed 166
6.2.4 Individual Post Comments Feed 166

6.2.5 Category and Tag Feeds 167
6.2.6 Other Feed Types 167

6.3.1 Feed Configurations & Formats 168

6.3.2 Full Feeds 170
6.3.3 Partial Feeds 171
6.3.4 Number of Posts 171
6.3.5 WordPress Feed Formats 172

6.4.1 Using FeedBurner For Feed Delivery 175

6.4.2 Benefits of Using FeedBurner 175
6.4.3 Setting Up & Configuring a FeedBurner Account 176
6.4.4 Redirecting to FeedBurner via Plugin 177
6.4.5 Redirecting to FeedBurner via HTAccess 177
6.4.6 Redirecting to FeedBurner via PHP 179

6.5.1 Tracking & Displaying Feed Statistics 181

6.5.2 Types of Statistics Provided by FeedBurner 181
6.5.3 Displaying FeedBurner Statistics 182
6.5.4 Alternatives to FeedBurner 183

6.6.1 Customizing Feeds 184

6.6.2 Formatting Feed Images 184
6.6.3 Adding a Custom Feed Image 187
6.6.4 Include Comments in Feeds 189
6.6.5 Creating Custom Feeds 191
6.6.6 More Feed Customization Tricks 195

6.6.7 Styling Feeds ... 196
6.6.8 Removing the WordPress Version Number ... 196
6.6.9 Disable and Redirect Unwanted Feed Formats ... 197
6.6.10 Insert Custom Content into Feeds ... 198
6.6.11 Importing and Displaying External Feeds ... 200
6.6.12 Buffer Period After Posting ... 200
6.6.13 Protecting Feed Content ... 202

6.7.1 Validating Feeds ... 203

7 Working with Comments

7.1.1 Optimizing the WordPress Comments Area ... 207
7.1.2 Welcome to the WordPress Comments Area ... 207
7.1.3 About the WordPress Comment System ... 208
7.1.4 Comments, Pingbacks, and Trackbacks ... 208
7.1.5 Anatomy of the WordPress Comment Area ... 209

7.2.1 Syndicating WordPress Comments ... 213
7.2.2 WordPress Main Comments Feed ... 213
7.2.3 Post-Specific Comment Feeds ... 214

7.3.1 Formatting the Comments Area ... 215
7.3.2 Using wp_list_comments() or a Custom Loop? ... 217
7.3.3 Implementing Paged Comments ... 223
7.3.4 Implementing Threaded Comments ... 225

7.3.5 Separating Comments, Pingbacks, and Trackbacks 228
7.3.6 Eliminating Pingbacks and Trackbacks 232
7.3.7 Control Comments, Pingbacks, and Trackbacks 234

7.4.1 Customizing Comment Display 236

7.4.2 Numbering Comments Globally and Locally 236
7.4.3 Alternating Comment Styles 240
7.4.4 Custom Styles for Authors & Members 241
7.4.5 Styling Comments with Gravatars 243
7.4.6 Add a "Your comment is awaiting moderation" Message 246
7.4.7 Moderation Links in the Theme Itself 247
7.4.8 Display Comment, Ping/Trackback Counts 248

7.5.1 Optimizing the Comment Form 249

7.5.2 Set up Comment Previews 249
7.5.3 Rich-Text Editors for Comments 250
7.5.4 Adding Comment Quicktags 252
7.5.5 Comment Management and Spam Prevention 254

7.6.1 Controlling Comment Spam 254

7.6.2 WordPress' Built-In Anti-Spam Functionality 255
7.6.3 Anti-Spam Plugins for WordPress 256

7.7.1 Other Considerations & Techniques 257

7.7.2 Enhancing and Encouraging Comments 257
7.7.3 Nofollow Links 258
7.7.4 Integrating Twitter 258

8 Search Engine Optimization

8.1.1 SEO Strengths and Weaknesses 261

8.1.2 Strong Focus on Content 261
8.1.3 Built-In "nofollow" Comment Links 262
8.1.4 Duplicate Content Issues 262

8.2.1 Controlling Duplicate Content 263

8.2.2 Meta noindex and nofollow Tags 264
8.2.3 Nofollow Attributes 267
8.2.4 Robots.txt Directives 269
8.2.5 Canonical Meta Tags 273
8.2.6 Use Excerpts for Posts 275

8.3.1 Optimizing Permalink Structure 276

8.3.2 Default URLs vs. "Pretty" Permalinks 276
8.3.3 Keep Permalinks Short 277
8.3.4 Maximize Permalink Keywords 280

8.4.1 Scoring with Google 281

8.4.2 Content, Content, Content 281
8.4.3 Detecting Duplicate Content 282
8.4.4 Optimizing Heading Elements 283
8.4.5 Optimizing Title Tags 284
8.4.6 The nofollow Wars 286
8.4.7 Fixing Broken Links 287
8.4.8 Using a Sitemap 288

8.4.9 SEO-Related plugins 289

8.5.1 Tracking the Success of Your Site 290

8.5.2 Statistical Plugins 290
8.5.3 Mint Stats 291
8.5.4 Google Analytics 292
8.5.5 Other Metrics 293
8.6.1 Closing Thoughts on SEO 294
Into the future... 295

9 Maintaining a Healthy Site

9.1.1 Keeping Your Site Healthy 297

9.1.2 Securing WordPress 297
9.1.3 Setting Secure File Permissions 298
9.1.4 Disabling Directory Views 300
9.1.5 Forbid Access to Sensitive Files 302
9.1.6 Remove the WordPress Version Number 313
9.1.7 Securing Your Database 313
9.1.8 Secure Multiple Installations 315
9.1.9 Prevent Hotlinking 315
9.1.10 More WordPress Security Help 317

9.2.1 Stopping Comment Spam 318

9.2.2 Configuring Your WordPress Admin Options 319
9.2.3 Using the Built-In Comment Moderation 320
9.2.4 Using the Built-In Comment Blacklist 320

9.2.5 Disabling Comments on Old Posts 321
9.2.6 Deny Access to No-Referrer Requests 321

9.3.1 Monitoring and Fixing Errors 322

9.3.2 Keep an eye on 404 Not Found Errors 323
9.3.3 Broken Link Checker Plugin 323
9.3.4 Other Error-Logging Techniques 324
9.3.5 Online Monitoring Services 324

9.4.1 Staying Current with WordPress 325

9.4.2 Updating WordPress 326
9.4.3 Logging Changes 327
9.4.4 Backing Up Your Database and Files 328

9.5.1 Optimizing WordPress 329

9.5.2 Content and File Caching 329
9.5.3 File Compression Methods 331
9.5.4 Optimizing CSS and JavaScript 332
9.5.5 Reducing the Number of HTTP Requests 333
9.5.6 Plugin Maintenance 337
9.5.7 Database Maintenance 338
9.5.8 Other Optimization Techniques 338
To Infinity, and Beyond 341
Thank you 342

Bonus Content

Log in to your account to download bonus content, exclusive themes, and more.

Life is conversational. Web design should be the same way. On the web, you're talking to someone you've probably never met - so it's important to be clear and precise. Thus, well structured navigation and content organization goes hand in hand with having a good conversation.

– CHIKEZIE EJIASI

3 Anatomy of a WordPress Theme

3.1.1 Understanding Theme Files

It is time for us to start taking a close look at how themes are built and how they work. If you have ever themed any kind of application before, you will appreciate how straightforward and intuitive WordPress theming actually is (with plenty of power when you need it). If you have never themed any application before, never fear, it's easy.

3.1.2 Every Theme is Different

Of course, the look of all themes is different. But if you were to download five different WordPress themes and open the folders side by side, you'll see a slightly different sets of files as well. There are a couple of required files and a number of files you will likely find in all themes, but beyond that the door is pretty wide open. For example, some themes might come with a special archives page because that theme is built to showcase archives in a unique way. Another theme might be missing `search.php`, because its `index.php` is built to handle search the functionality.

3.1.3 Commonly Used Theme Files

In the chart on the next page, notice how we have labeled each of the theme files. Two of them, `index.php` and `style.css` are labeled as "CORE". This means that they are absolutely essential to your theme. In fact, WordPress will not recognize any theme if these two files are not within the theme folder. Technically, you

STANDARD (used in most themes)

SPECIAL (optional additions)

CORE (required)

JUNK (legacy, don't use)

404.php	Error page, served up when someone goes to a URL on your site that doesn't exist
archive.php	Page that displays posts in one particular day, month, year, category, tag, or author
archives.php	Page template that includes search form, category list, and monthly archives (requires page using it)
comments-popup.php	If you enable popup comments (obscure function), the comments link will use this template
comments.php	This file delivers all the comments, pingbacks, trackbacks, and the comment form when called
footer.php	Included at the bottom of every page. Closes off all sections. (Copyright, analytics, etc)
front-page.php	Displays content for the site's front page, aka home page
functions.php	File to include special behavior for your theme.
header.php	Included at the top of every page. (DOCTYPE, head section, navigation, etc)
home.php	Displays content for the site's front page if front-page.php is not available
image.php	If you wish to have unique pages for each of the images on your site (for credits, copyright…)
index.php	This is typically the "homepage" of your blog, but also the default should any other views be missing
links.php	Special page template for a home for your blogroll
loop.php	Common in newer themes, an optional file to house your custom, multiple, or regular loops
page.php	Template for Pages, the WordPress version of static-style/non-blog content
rtl.css	A special CSS file for your optional inclusion to accommodate "right to left" languages
screenshot.png	This is the image thumbnail of your theme, for help distinguishing it in the Appearance picker
search.php	The search results page template
sidebar.php	Included on pages where/when/if you want a sidebar
single.php	This file is displays a single Post in full (the Posts permalink), typically with comments
style.css	The styling information for your theme, required for your theme to work, even if you don't use it

could build a theme with only these two files. And a simple theme it would be! That might be just what you need for some special applications of WordPress, but in general, you are probably using WordPress because you want a bit more functionality than that would offer.

Most themes will include both the CORE and "STANDARD" files. The STANDARD files cover everything both you and your visitors will expect from a blog. Things like permalinked posts and pages, error catching, commenting, and organized archives.

Some of these files are marked as "SPECIAL", in that they offer something above and beyond the basics. For example, the `image.php` file. If you choose to use the WordPress default media library to manage the files you post to your site (images, movies, etc.), you can insert them into your posts with a link to a special page on your site controlled by the `image.php` file. This can be useful. You can include special information on this page like copyright information, author information, usage rights, etc. Not all sites would want or need this, hence its designation as SPECIAL.

One of the files listed, `comments-popup.php`, is marked as "JUNK" because it's no longer used but still found in many themes. The file is just weird; we could tell you all about it, but it's not worth the ink (really).

More to the list

The chart on the opposite page isn't a complete list of every template file, just the most common ones (see page 57 for more). You may create as many of your own custom theme templates as you like. Get more details at the WP Codex:

https://digwp.com/u/634

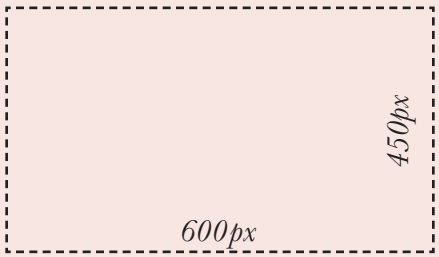
Singular Template

You can use a `singular.php` *template to display both single posts and pages. Kinda like combining* `single.php` *with* `page.php`*:* `singular.php`

Brand Your Theme

Adding a screenshot and description to your theme is a great way to give it an distinct identity. Here's how:

1 *Create a file named* `screenshot.png` *and put it in your theme folder.*



2 *Put this description at the top of your* `style.css` *file*

```
/*
Theme Name: Theme Name
Theme URI: https://your-website.com/cool-theme/
Description: Totally awesome WordPress theme by
<a href="https://your-website.com/">Yours Truly</a>
Version: 1 (WP2.8.4)
Author: Your Name
Author URI: https://your-website.com/
Tags: super, awesome, cool, sweet, potato nuggets
*/
```

3.1.4 How Theme Files Work Together

These files are not stand-alone templates. They interact and call upon each other to get the job done. For example, `index.php` alone will call and insert `header.php` at the top of it, `sidebar.php` in the middle of it, and `footer.php` at the bottom of it. Then, the `sidebar.php` file might have a function to call in `searchform.php`. Likewise, the `header.php` file, which includes the `<head>` section, will call upon the `style.css` file.

It is this modular, dynamic approach that gives WordPress theme building a lot of its power. For those folks coming from a background of building static sites, the nature of using templates is probably already quite appealing. Imagine wanting to add a navigational item to the site's main menu bar, which likely lives in the `header.php` file. One change, and the new navigational item is reflected on all pages of the site. Going further, the menu bar itself is likely generated from a built-in WordPress function. As soon as you publish a new page from the Admin area of WordPress, the menu-bar function will recognize the new page and automatically append it to the sitewide menu bar. This is powerful stuff that makes site modifications, updates, and management very easy.

Home Sweet Home

Confused about the difference between `home.php` *and* `front-page.php`*? This post should help:*

https://digwp.com/u/671

Similarly confusing are the template tags, `home_url()` *and* `site_url()`*, as explained here:*

https://digwp.com/u/672

3.2.1 Understanding Different Page Views

There are really only a handful of different types of page views:

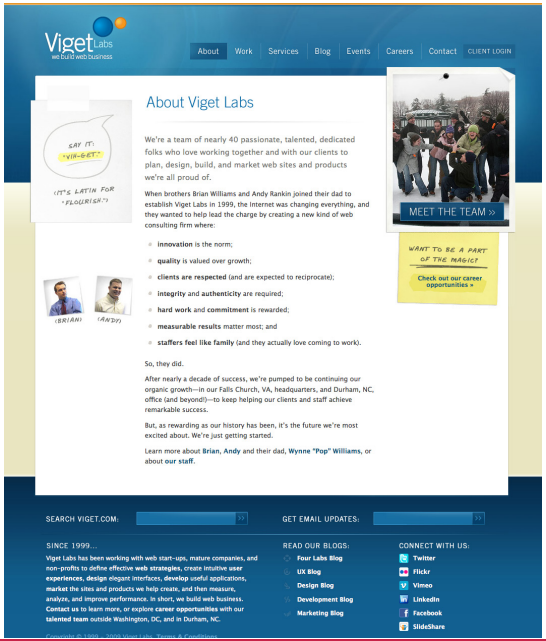
- **The Home Page -** usually at the root URL of your domain
- **Single Posts -** displays one post at a time, usually in its entirety
- **Static Pages -** pages that are outside the flow of normal posts
- **Custom Pages -** static pages that have been customized
- **Search Results -** displays a list or summary of posts matching a search
- **Archive -** shows series of posts for categories, tags, dates, and authors

3.2.2 Page Views are for Pages

We already learned about Pages and how they are most commonly used for "static" style content. You cannot categorize or tag a Page, they exist outside the chronological flow of posts, and they don't appear in the RSS feed like Posts do. As such, the theme template used to display Pages is generally different than that used to display Posts. For example, it may lack the functionality to display things such as dates, author names, and comments. Instead, it might include functionality to display the breadcrumb trail of its hierarchy of parent pages (see chapter 5.5.6).

3.2.3 Single Views are for Posts

The `single.php` file is responsible for displaying a single Post. There may be parts of the `single.php` template file for displaying categorization and other "meta" information about the post, as well as the functionality required for displaying the comments area and comment form. Perhaps you want your single posts to be a bit wider and less cluttered? The `single.php` file is where you might omit calling the sidebar and adjust your CSS accordingly.



PAGE

Regular Title

No comments
This content isn't really meant for public discussion.

Unique sidebars
The sidebar needs on this page are different than elsewhere on the site. WordPress can accommodate.

Nav Highlighting
About page = About highlighted in navigation

POST

Extra Blog Header
Blog posts have "blog" header in addition to title and meta about this post.

Comments
This content is meant for public discussion. (not visible in screenshot, but there!)

Unique sidebars
Blog area has blog-related ancillary content, like categories, subscription info, and popular content.

Nav Highlighting
Any blog page = Blog highlighted in navigation

3.2.4 The Many Faces of Archive Views

There are many types of archives, and this one file, `archive.php`, is often in charge of displaying them all. When viewing a particular category, tag, author, or date-based archive, WordPress will generate the markup and display the content according to the code contained in the `archive.php` file.

Look at all the archive links at the Digging Into WordPress site. Every one of those subsequent pages is handled by the `archive.php` file

3.2.5 How WordPress Decides Which File to Use for Rendering the View

All this talk about different page views is begging the question, "how does WordPress figure out which template file to use?" You might assume that it is hard-wired into WordPress, but as we've learned, most of the files in a theme are optional. If your theme doesn't have an `archive.php` file, does WordPress just display a blank page? Absolutely not, it moves down its hierarchy of template files to find the next most appropriate file to use. Ultimately, all paths in the WordPress templating world end at the `index.php` file. No wonder this is such an important and required file!

Just as we move down the hierarchy toward `index.php`, we can travel in the other direction and create template files that are very specific. For example, if we wish to have a unique template when viewing category #456 of our blog, we can create a file called `category-456.php`, and WordPress will automatically use it. Let's take a look at the hierarchy flowchart.

WHICH TEMPLATE FILE WILL WORDPRESS USE?

THE TEMPLATE HIERARCHY

PAGE TYPE	tries first > tries next > tries last
404	404.php > index.php
SEARCH	search.php > index.php
TAXONOMY	taxonomy-{tax}-{term}.php > taxonomy-{tax}.php > taxonomy.php > archive.php > index.php
HOME	home.php > index.php
ATTACHMENT	{mime-type}.php > attachment.php > single.php > index.php
SINGLE	single-{post-type}.php > single.php > index.php
PAGE	{custom-template}.php > page-{slug}.php > page-{id}.php > page.php > index.php
CATEGORY	category-{slug}.php > category-{id}.php > category.php > archive.php > index.php
TAG	tag-{slug}.php > tag-{id}.php > tag.php > archive.php > index.php
AUTHOR	author-{author-nicename}.php > author-{author-id}.php > author.php > archive.php > index.php
DATE	date.php > archive.php > index.php
ARCHIVE	archive.php > index.php

3.3.1 Kicking It Off with the Header

If you had never seen the files in a WordPress theme before, you could probably guess which file is responsible for the top of pages. It's everybody's favorite theme file: `header.php`!

3.3.2 The DOCTYPE and HTML Attributes

In 99.999% of all themes, the header file is the *first* file that is called when WordPress is rendering any type of web page. As such, its contents begin with the same code that *all* web pages begin with, namely, the DOCTYPE. This isn't the time or place to talk about why to choose one DOCTYPE over another (there are plenty available to choose from). Just suffice it to say that HTML 5 is a very common DOCTYPE choice these days. Here's how it looks in the source code:

```
<!DOCTYPE html>
```

Directly after any site's DOCTYPE element is the opening HTML tag, which may include attributes that work with the DOCTYPE to prepare the browser for what to expect from the source code. A commonly used attribute for the `<html>` tag is `lang`, which specifies the page's language. At this point, WordPress jumps in with a template tag to help define the page's language attributes:

```
<html <?php language_attributes(); ?>>
```

So putting those two lines together gives us something very simple and elegant:

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
```

It just doesn't get much better than that.

3.3.3 META Elements

After the opening `<html>` tag, we move into the `<head>`, which is also common to all web pages and provides all sorts of information the browser needs to display the page as intended. Within the `<head>` section, we begin with some choice `<meta>` tags, which can be thought of as "information about information." In this case, the HTML is the information, and so meta tags describe that information. To let the browser know the content type and language used, WordPress helps us with some super-handy template tags:

```
<meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>;
    charset=<?php bloginfo('charset'); ?>">
<meta charset="<?php bloginfo('charset'); ?>">
```

Simplified HTML5

Here are two similar ways of declaring the character set. In HTML5, these tags are equivalent. So use the shorter one to keep things simple.

Other important meta tags include "description" (very important) and "keywords" (less important). But because the description and keywords for any given page on your site depend on the actual content of that page, it is best to dynamically generate these tags rather than include them directly here. See page 47 for information about the All in One SEO plugin to handle meta tags automatically.

3.3.4 The Title

The `<head>` is also where the `<title>` for the page is declared, which is an incredibly important line in any HTML code. It is literally what is shown at the top of the browser window, what is saved as the default title of bookmarks (both saving locally and socially), and is used for the title link in search-engine listings. Again, we are in the tough position where this bit of code is written only once, right here, and is used for every single page on the entire site. So how do you craft it so that the title is optimal on every possible page? Glad you asked.

Here is an excellent function that enables top-notch, attractive-looking and descriptive titles for every possible type of web page. Simply use this code as the `<title>` element in your theme's `header.php` file and you're good to go:

New Title Tag Technique

WordPress has a new way to implement title tags. Check out the WP Codex to learn more:

https://digwp.com/u/676

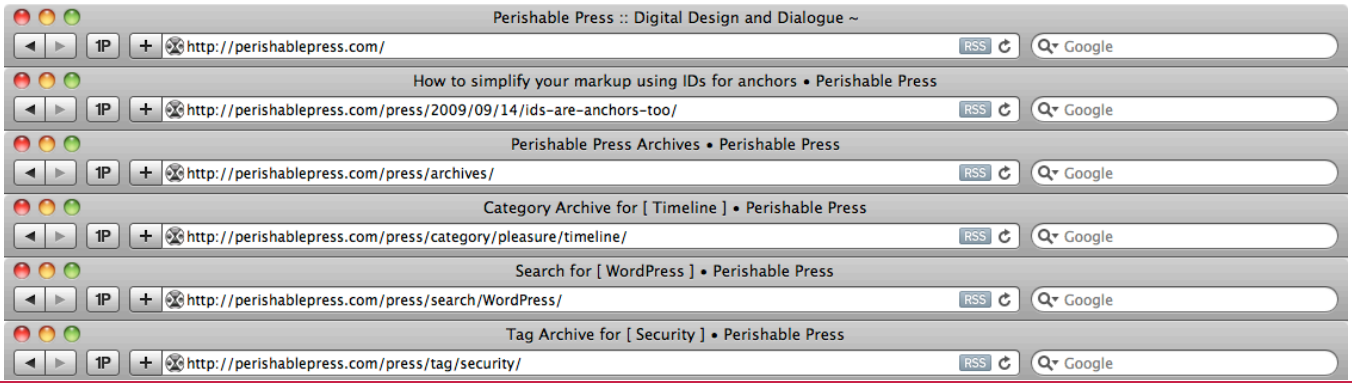
Perfect Title Tags

For the full scoop on creating perfect title tags for your WordPress-powered site, check out these two articles:

https://digwp.com/u/397
https://digwp.com/u/398

```
<title>
    <?php if (function_exists('is_tag') && is_tag()) {
          single_tag_title('Tag Archive for &quot;'); echo '&quot; - ';
    } elseif (is_archive()) {
          wp_title(''); echo ' Archive - ';
    } elseif (is_search()) {
          echo 'Search for &quot;'.wp_specialchars($s).'&quot; - ';
    } elseif (!(is_404()) && (is_single()) || (is_page())) {
          wp_title(''); echo ' - ';
    } elseif (is_404()) {
          echo 'Not Found - ';
    }
    if (is_home()) {
          bloginfo('name'); echo ' - '; bloginfo('description');
    } else {
          bloginfo('name');
    }
    if ($paged > 1) {
          echo ' - page '. $paged;
    } ?>
</title>
```

Those sure would bookmark nicely, wouldn't they?



The All-In-One SEO Plugin that we mentioned earlier can also be put in charge of handling page titles. The advantage is that it keeps this area of the theme cleaner and does provide what is generally considered the best page title format for SEO. The disadvantage being that it isn't very customizable or nearly as configurable as doing it yourself.

3.3.5 Link Elements

The `<head>` is also the place to link to external resources like CSS and JavaScript files. Since your theme requires the presence of a `style.css` file in the root directory of your theme, you might as well use it. Including it is as simple as this:

```
<link rel='stylesheet' href='<?php bloginfo("stylesheet_url"); ?>'
    type='text/css' media='screen'>
```

Parameterized is a fun word, isn't it?

The parameterized function, `bloginfo("stylesheet_url")`, literally returns the exact URL of the stylesheet. No reason to hard-code anything here. And in fact, the `bloginfo()` function can return all sorts of useful information, which we'll dig into shortly.

On the other hand, including JavaScript files in your theme is slightly trickier, especially if you want to do it the *right* way (you do). Let's say you want to include the popular JavaScript library jQuery on your page, and also a custom script of your own that makes use of jQuery. Because jQuery is such a popular library, it is used fairly commonly by other plugins, and in fact by the WordPress Admin area itself. As such, WordPress literally ships with a copy of jQuery you can link to. To do so, simply call this function in your head area or `functions.php` file:

```
<?php wp_enqueue_script('jquery'); ?>
```

Doing it this way has a few distinct advantages.

1. **It's easy.** It creates a link to a file you know is there and you know works.
2. **It lets WordPress know** that the requested file is successfully loaded.

jQuery

https://jquery.com/

If you go off and download your own copy of jQuery and link to that, WordPress has no idea that you've done this. Then if you start using a plugin that utilizes jQuery, it will go off and load another copy, which will cause all sorts of havoc. Conversely, if you enqueue the file instead, the plugin will recognize the fact it already exists and not load a duplicate copy. *Hurrah!*

On the other hand, when you load your *own* script, you don't really need to enqueue it because it is already totally unique and not included in WordPress. You can load your own script on the page like this:

```
<script type="text/javascript"
    src="<?php bloginfo('template_url'); ?>/js/myscript.js"></script>
```

As you can see, we are using another `bloginfo` function here, only this time it outputs the URL path to the active theme directory, not to any particular file.

Now, let's say on your archives pages that you have a whole bunch of special CSS that isn't used anywhere else on the site and a custom script that is unique to your archives pages. You can use some special WordPress logic to detect if the archives pages are the ones being viewed, and load the files only in that situation:

```
<?php if (is_page_template('page-archives.php')) { ?>
    <link rel="stylesheet" href="<?php bloginfo('template_url'); ?>/css/
archives.css" type="text/css" media="screen">
    <script type="text/javascript" src="<?php bloginfo('template_url'); ?>/
js/archives.js"></script>
<?php } ?>
```

That will take effect if you are using a special page template for your archives that is literally named "`page-archives.php`". If instead you happen to know the ID of the page (available in the Admin area, see note on next page), that could be written like this:

```
<?php if (is_page("5")) { ?>
    <link rel="stylesheet" href="<?php bloginfo('template_url'); ?>/css/
archives.css" type="text/css" media="screen">
    <script type="text/javascript" src="<?php bloginfo('template_url'); ?>/
js/archives.js"></script>
<?php } ?>
```

…where "5" in the first line is the page ID. Feel free to use PHP's "or" operators here to cover multiple pages.

Putting all of that together, our code looks something like this:

```
<?php wp_enqueue_script('jquery'); ?>
<?php wp_head(); ?>
<script type="text/javascript" src="<?php bloginfo('template_url'); ?>/js/
myscript.js"></script>
<?php if (is_page("5")) { ?>
    <link rel="stylesheet" href="<?php bloginfo('template_url'); ?>/css/
archives.css" type="text/css" media="screen">
    <script type="text/javascript" src="<?php bloginfo('template_url'); ?>/
js/archives.js"></script>
<?php } ?>
```

Hey! What's up with that wp_head() thing? Glad you asked…

What is My Page ID?

Determining the ID of your posts and pages is not as easy as it used to be. In previous versions of WordPress, the ID was conveniently displayed right next to the post or page in the Admin area.

In newer versions of WordPress, ID information has been removed, and is only accessible by hovering over the post/page link in the Admin's Edit Posts or Edit Pages screens.

Thus, to get your ID, hover over its link in the Admin and look at your browser's Status Bar to see the information. It will be appended to the URL as the last parameter value.

3.3.6 The wp_head() Function

A must for any theme, the `wp_head()` function simply tells WordPress "Right here, this is inside the `<head>`." It is kind of a generic function that is used as a "hook" on which the WordPress core, plugins, and custom functions may attach things.

For example, if you have the XML-RPC functionality of your blog enabled (Settings > Writing), it requires a special `<link>` element to be added into the `<head>`. If it is present within your theme, the `wp_head` function will be used by WordPress to include the required XML-RPC element to the `<head>`.

Similarly, in the previous section, the code uses the `wp_enqueue_script` function. All by itself, that function doesn't have any effect. But when the `wp_head` tag is also present, it serves as a hook that serves as the location at which the `wp_enqueue_script` function will load the script.

Plugins also use the `wp_head` function to load their own scripts and CSS files. Sometimes they even insert inline CSS and JavaScript, which is a bit annoying and makes for a messy "View Source" experience.

3.3.7 Template Tags

Now is a good time to mention that there is a WordPress function for pulling out a variety of information about your blog. This information is useful on a regular basis when creating themes. Here is the function…

```
<?php bloginfo('template_url'); ?>
```

…and here is the different types of data that you can get from it:

```
admin_email = jeff@digwp.com
atom_url = https://digwp.com/home/feed/atom
charset = UTF-8
```

```
comments_atom_url = https://digwp.com/home/comments/feed/atom
comments_rss2_url = https://digwp.com/home/comments/feed
description = Take Your WordPress Skills to the Next Level!
url = https://digwp.com/home
html_type = text/html
language = en-US
name = Digging Into WordPress
pingback_url = https://example/home/wp/xmlrpc.php
rdf_url = https://digwp.com/home/feed/rdf
rss2_url = https://digwp.com/home/feed
rss_url = https://digwp.com/home/feed/rss
siteurl = https://digwp.com/home
stylesheet_directory = https://digwp.com/home/wp/wp-content/themes/largo
stylesheet_url = https://digwp.com/home/wp/wp-content/themes/largo/style.css
template_directory = https://digwp.com/home/wp/wp-content/themes/largo
template_url = https://digwp.com/home/wp/wp-content/themes/largo
text_direction = ltr
version = 2.8.5
wpurl = https://digwp.com/home/wp
```

If you were looking closely, you may have noticed we have already used this function earlier in our example showing how to include a stylesheet:

```
<link rel="stylesheet" href="<?php bloginfo('template_url'); ?>/css/
archives.css" type="text/css" media="screen">
```

This is how you can generate a URL from inside your theme folder without having to hard-code anything or worry about relative file paths. Hard-coding is problematic (what if you change the name of the theme?). Relative file paths are problematic too, because the URL structure of a site can change and go many levels deep, the only reliable way to do it is to start with the root (“/”), which would then require the theme’s folder name anyway.

Global Custom Fields

Another way to look at the `bloginfo()` function (see 3.3.7) is as a "Global Custom Field." That is, a value that you can access from anywhere that returns a value you can use. Posts and Pages can have custom fields as well, but they are localized to that Post or Page and thus not very "Global." Creating your own global custom fields could potentially be very useful. For example, let's say you use the Amazon Affiliate Program to help your site earn money. This affiliate code is baked into all sorts of data that you can get from Amazon, like URLs for linking to products and their widgets. As with everything, you could hard-code this affiliate code everywhere it needs to be, but that isn't a very efficient technique. If this code were to change some day (you never know), you are stuck updating a lot of code. Instead, let's do it right by literally creating a custom settings area in the Admin for creating our own global custom fields.

Add this to your `functions.php` file:

```
<?php add_action('admin_menu', 'add_gcf_interface');

function add_gcf_interface() {
    add_options_page('Global Custom Fields', 'Global Custom Fields', '8', 'functions',
    'editglobalcustomfields');
}

function editglobalcustomfields() { ?>
    <div class="wrap">
        <h2>Global Custom Fields</h2>
        <form method="post" action="options.php">
        <?php wp_nonce_field('update-options') ?>
            <p><strong>Amazon ID:</strong><br>
                <input type="text" name="amazonid" size="45"
                value="<?php echo get_option('amazonid'); ?>">
            </p>
            <p><input type="submit" name="Submit" value="Update Options"></p>
            <input type="hidden" name="action" value="update">
            <input type="hidden" name="page_options" value="amazonid">
        </form>
    </div>
<?php } ?>
```

You can now display this value anywhere in your theme with the `get_option()` template tag:

```
<?php echo get_option('amazonid'); ?>
```

GCF @ DigWP.com

A bit more about Global Custom Fields at DigWP.com:

https://digwp.com/u/586

https://digwp.com/u/587

3.4.1 The WordPress Loop

The loop is the *one thing* that is *absolutely core* to understanding how WordPress works. In its most basic, generalized form, the loop looks like this:

Bad analogy!
Bad analogy!

```
<?php
// The Loop
if (have_posts()) : while (have_posts()) : the_post();
...
endwhile; else:
...
endif;
?>
```

As veteran developers know, a "`while`" loop is a standard concept in any programming language, and its use here is just standard PHP. First the loop makes sure that there are some posts to display (the "`if`" statement). If that is true, it begins the loop. Then, the function "`the_post()`" sets the stage for WordPress to use inner-loop functions, which we will explore soon. Once `the_post()` has been iterated the specified number of times, "`have_posts()`" turns to false and the loop stops.

Yikes! That is sounding pretty abstract. Perhaps we better break things down so we don't lose each other.

3.4.2 The Loop in Plain English

Are there any posts published? Sorry, just had to ask, the rest of this code will go funky if there aren't any.

Begin the loop. This will cycle through the number of Posts you have set to display (under **Settings > Reading**).

A header tag with an anchor link inside it. The text will be the title of the Post, and the link will be the permalink to the single Post page.

A custom field that is attached to this Post is pulled out and displayed. In this case, the key of "PostThumb", which returns an "<img>" tag symbolizing this Post.

"Meta" information about the Post is displayed: the Month Day, Year the Post was published and the display name of the Author who wrote it.

The full content of the Post is displayed.

More meta information about the post is displayed: all the tags and categories given to this Post and the number of comments, which is a link to the commenting area.

End of the loop

If there are older or newer posts available, display links to them.

No posts? (a failsafe)

Better tell the people.

All done.

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

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

    <div class="post" id="post-<?php the_ID(); ?>">

     <h2><a href="<?php the_permalink(); ?>"
rel="bookmark" title="Permanent Link to <?php the_
title_attribute(); ?>"><?php the_title(); ?></a></h2>

     <?php echo get_post_meta($post->ID, 'PostThumb',
           true); ?>

     <p class="meta">

       <span>Posted on</span> <?php the_time('F jS,
       Y'); ?> <span>by</span> <?php the_author(); ?>

     </p>

     <?php the_content('Read Full Article'); ?>

     <p><?php the_tags('Tags: ', ', ', '<br>'); ?>
     Posted in <?php the_category(', '); ?>
     <?php comments_popup_link('No Comments;',
     '1 Comment', '% Comments'); ?></p>

   </div>

 <?php endwhile; ?>

 <?php next_posts_link('Older Entries'); ?>

 <?php previous_posts_link('Newer Entries'); ?>

<?php else : ?>

 <h2>Nothing Found</h2>

<?php endif; ?>
```

3.4.3 The Loop Just Knows…

As mentioned, the loop is simply a dressed-up "`while`" loop. While there are posts available in the database, display the posts. In theory, it's simple and utilitarian. But what might remain confusing is just how this `while` loops knows exactly what to loop. While… what? Well, without you having to tell it, the basic loop function already knows what its query is going to be! To see for yourself what the query string is, you can echo it to the web page by adding this little snippet directly before the loop:

```
<?php global $query_string; echo $query_string; ?>
```

If we were to place this snippet above our `single.php` loop at the Digging Into WordPress site, the following information would be displayed on any single page:

```
year=2011&monthnum=02&name=version-3-update
```

In plain English, that reads: "The date is February 2011 and the post name is *Version 3 Update*." Likewise, if we echo that `$query_string` variable from our `archive.php` file, and then visit the "JavaScript" category archive, we see this:

```
posts_per_page=10&what_to_show=posts&orderby=date&order=DESC&category_
name=javascript
```

In plain English: "Show ten Posts from the Javascript category in descending chronological order."

Note that we did nothing *manually* to change this query string, but merely by loading a different type of page (an archive view), WordPress provides the proper query to make that loop do the right thing. Don't worry if this sounds confusingly technical. It doesn't really matter. The point is that The Loop *just knows* what to loop through for the type of page you are building and displaying.

loop.php

Many WP themes include a `loop.php` *file, which helps reduce repetitive code in other theme files. Explore!*

Want to keep going?

Get the book!

https://digwp.com/book/