

ERHVERVSAKADEMI AARHUS  
BUSINESS ACADEMY AARHUS

# WORDPRESS IN THE CLASSROOM

By Per Thykjaer Jensen



Published by Business Academy Aarhus, Research and Innovation department.

## **WORDPRESS IN THE CLASSROOM – PART ONE**

Published by Business Academy Aarhus,  
The Research and Innovation department, August 2017.

### **Research and innovation publication #7**

*We work with applied research, development, and innovation  
that create value for educational programmes, companies and the society.*

*Read more about our research and innovation projects:  
<http://www.eaaa.dk/forskning-og-innovation/projekter/>*

*The publication is published as a part of the research project  
"What You Should Know About WordPress".  
Research blog: [research-wordpress.dk/](http://research-wordpress.dk/)*

### **TEXT AND CONCEPT**

Per Thykjaer Jensen, Senior Lecturer and Project Manager,  
Business Academy Aarhus,  
The Research and Innovation department.

Contact: [pertj@baaa.dk](mailto:pertj@baaa.dk), tel: +45 7228 6321. Twitter: @pertjensen  
URL: <http://research-wordpress.dk/>

### **EDITORS**

Ulla Haahr and Karina Hansen,  
Business Academy Aarhus,  
Research and Innovation department.

### **PROOF READERS**

Bror Arnfast, Senior Lecturer, Business Academy Aarhus.  
Mark Hughes, Senior Lecturer, Business Academy Aarhus.

### **LAYOUT**

René Kristensen, Business Academy Aarhus.

### **ISBN**

978-87-999767-2-0

### **COPYRIGHT**

Creative Commons License: Attribution-NonCommercial-ShareAlike  
4.0 International



# Contents

Part one:

## **Making WordPress**

<b>What You Should Know About WordPress</b>	<b>12</b>
<b>The History of WordPress</b>	<b>14</b>
<b>The Core-Themes</b>	<b>16</b>
<b>Themes: 2010 – 2017</b>	<b>19</b>
<b>Themes as Visions for WordPress</b>	<b>30</b>
<b>Anthropology and Open Source</b>	<b>32</b>
<b>Artefacts</b>	<b>33</b>
<b>Espoused Values</b>	<b>35</b>
<b>Basic Assumptions</b>	<b>39</b>
<b>The Prelude on Github</b>	<b>43</b>
<b>Schein, Organization and Development</b>	<b>48</b>



## Part two:

# Diving Into Themes

<b>The Theme-code</b>	<b>52</b>	<b>Partial Conclusion</b>	<b>78</b>
<b>Theme Building Blocks</b>	<b>53</b>	<b>Why you need a child theme</b>	<b>80</b>
<b>Three important files</b>	<b>54</b>	<b>How to make a Child Theme</b>	<b>81</b>
<b>The style.css</b>	<b>55</b>	<b>How to activate jQuery</b>	<b>84</b>
<b>The index.php</b>	<b>57</b>	<b>Nodejs and WordPress</b>	<b>86</b>
<b>header.php and footer.php</b>	<b>60</b>	<b>GULP and SASS</b>	<b>90</b>
<b>functions.php</b>	<b>62</b>	<b>Innovations in WordPress 4.7.x</b>	<b>96</b>
<b>Menus</b>	<b>66</b>	<b>How to use the REST API</b>	<b>98</b>
<b>Theme Widgetizing</b>	<b>70</b>	<b>Introduction to the wp-cli</b>	<b>100</b>
<b>Advanced Loops: Snippets and Tags</b>	<b>73</b>	<b>Install wp-cli</b>	<b>103</b>
<b>The Template Hierarchy</b>	<b>76</b>	<b>Final Words</b>	<b>106</b>
		<b>References</b>	<b>108</b>

# Introduction



Per Thykjær Jensen



Project Manager and Senior Lecturer

Business Academy Aarhus  
Research and Innovation

Phone: +45 7228 6321  
petj@eaaa.dk

On the internet, about one in four pages are “powered by WordPress”. “WordPress in the Classroom” is in two parts. The first part will give you knowledge about WordPress, and the second is an introduction to WordPress frontend development:

- **Part One: Making WordPress**

Here you will find an introduction to the history and the open source philosophy behind WordPress. Part one will give an idea of the creative process behind the Twenty Seventeen theme. This part is an answer to the question: What should the upcoming WordPress professional know in order to meet the demands of business?

- **Part Two: Diving into Themes**

The second part focuses on WordPress frontend development. You will learn how to create your own design - either from scratch or tweak an existing theme by a child theme. Part two will also introduce some new trends in WordPress frontend development such as Nodejs features and command line tools.

The book is the output from a research project at the Research and Innovation department at Business Academy Aarhus. The project was initiated by an increasing number of students used WordPress in their projects, or having to work with WordPress during internships.

This observation led to the research question: what should the multimedia student know about WordPress in order to meet the demands of future

employers? Several months of study followed. Books on WordPress, wikis and research articles were consulted. Business professionals and students were interviewed.

Here you can see how WordPress emerge from the creative vision of the open source community behind WordPress. Most researchers focus on the usability of WordPress. Most of the articles I found were studies of a particular WordPress solution. The researchers tested the solution on a number of users. By statistic method the authors would conclude whether WordPress was usable or not.

An empiric method like that will not explain how or why certain WordPress features are made. And since WordPress is of vital importance for a great variety of businesses, such knowledge is important. That's why I decided to follow the making of a new

theme. I guess that most people will be able to follow part one of the book, since it's a narrative.

I also followed the work of the core developers closely, and came to know more about their workflow, methods and code. The second part of the book focuses on front-end development code. That is: how to create a WordPress theme – or web design so to speak.

#### If you teach, “WordPress in the Classroom” can give you:

- Insight into the workflow and know-how of WordPress professionals.
- The history and the creative process behind the creation of WordPress.
- What you need to know in order to create a stunning WordPress theme.
- Innovations in WordPress 4.7.x

If you have a basic knowledge of HTML, CSS, JavaScript and possibly PHP, you will be able to follow the instructions given in the second part.

“WordPress in the Classroom” is intended for those who need to either teach or study WordPress. I hope that it will open the doors, and ease the path for the upcoming professional open source developer.

#### In you're studying and expect to use WordPress in your professional career this book will give you:

- Information about the open source philosophy behind WordPress
- Knowledge about how to make a WordPress theme.
- What's new or coming up? New trends in WordPress.

A classroom is basically like a lab. Here we can experiment. Try stuff, and work with background information in ways that may be possible in real life working situations. In real life situations, you just have to deliver. The idea behind this book is to open the gates to WordPress. What is it. How you can develop solutions, and to see how WordPress is used by small and medium sized businesses.

“WordPress in the Classroom” is meant as a handbook that will open doors to a deeper understanding of WordPress. These days, WordPress is the content manager behind roughly 25% of all web pages. Understanding WordPress is vital for anyone studying the web - or working as a professional web developer.

The book was prepared for WordPress professionals in small to medium sized businesses. It will answer questions like:

- How do WordPress professionals work with the CMS in small to medium sized businesses?
- And what should they know in order to be WordPress practitioners?

Most WordPress users take the software more or less for granted. If a business depends on WordPress, it is important to know how WordPress is *made*. During the preparations for the book, I assumed the role of a WordPress maker and followed the development of the *Twenty Seventeen* WordPress-core theme. The process gave valuable knowledge about the creative process behind WordPress. If you should want to contribute to WordPress,

the book will give you valuable information.

Open source is actually organized in a very professional way. In the book, you can follow how a core-theme was made. You will get valuable information about developing your own themes.

“WordPress in the Classroom” is not meant as a tutorial, nor is it an omniscient manual. The book is more like a Swiss army knife for WordPress teachers and students. The book will point out where you can find the knowledge you need in order to work as a WordPress professional. Several code samples were made. All code is available via Github. The main code repositories are:

- A primitive theme, called [Petj-mini-theme](#). I guess that this theme is almost as minimal as possible.

- A Bootstrap theme, called [Bootstrap-F16-Skeleton](#). Here the code demonstrates how to make a theme that's ready for Bootstrap and even jQuery.
- Yet another theme skeleton, called: [Petj-mvp](#)
- Finally, the theme [Nuit](#). It is a theme with Nodejs features.

It is possible to create very good WordPress solutions with very little or even no code at all. A WordPress professional should have at least some understanding of the mechanisms of a theme. You don't have to be an expert in order to read the code, but a little knowledge about HTML, CSS and JavaScript is necessary.

I hope that you will enjoy "WordPress in the Classroom".



# Part one: **Making WordPress**

Here you will find an introduction to the history and the open source philosophy behind WordPress. Part one will give an idea of the creative process behind the Twenty Seventeen theme. This part is an answer to the question: What should the upcoming WordPress professional know in order to meet the demands of the business?





# What You Should Know About WordPress

---

There is a clear difference between using WordPress in the business world and in the classroom. In business life, one just has to deliver. There is a customer, and the developer must comply with whatever demands the client may have. As a contrast, the classroom is a lab where experiments are possible.

Interviews with a student and a professional web designer gave valuable information about WordPress in the business world. To my surprise, knowledge about plugins and web shops is an important competence. So is code. You simply have to develop a multimedia product that is useful to the client.

## Case I: The student working for a real life client

"In my experience, there are two days of headache where you have to figure out how everything works, how these plugins work, and why they don't behave as I want them to. That is until I understand. I don't know why, but at a certain time everything just works at a higher level." (Sandra Kristholm quoted in Jensen 2016b).

As a Multimedia student, Sandra Kristholm developed a WordPress CMS for a business client, and wrote her final dissertation about the case. Sandra developed a useful solution for a local organization in Aarhus. Most of the functionality on the web site was added via plugins and the theme's tweaking options. She produced a multimedia solution that would work out of the box. (Kristholm, 2016).

- "If time is a problem you should go for WordPress," Sandra stated.

## Case II: The WordPress business professional

Daniel Pape is a WordPress professional. After finishing his education at Aarhus Business Academy, he was employed at the company Erhvervshjemmesider. The company creates web sites for businesses. Daniel told me about his experience as a WordPress developer.

"WordPress is not perceived as "hard-core" web development", Daniel stated. Probably, there is some truth in this observation. Once a fellow developer asked him

- "Is it really true, that you sell WordPress solutions?"  
- "Yea, WordPress is super easy for us and for the customers," Daniel answered.

Actually, the critique from the fellow coder misses the target. For most clients even WordPress is considered as something, where you really need an expert. Only a few clients are able to develop a web site, that is to tweak a site with a little CSS or JavaScript. That's why a WordPress webpage is a product you can sell.

Daniel's advice is to get thorough knowledge of one theme. It should be a theme with many options for personalization and tweaks. In Daniel's opinion, knowing WooCommerce is necessary:

"A multimedia designer should know WooCommerce. There's a big market out there for web shops. If a multimedia designer needs freelance jobs there are many customers who want a web shop." (Daniel Pape quoted in Jensen 2016a).

### What you should know about WordPress?

First of all the WordPress expert should know her clients, and create useful solutions that are easy to use. Often this solution is a creative collage of plugins for web shops, search engine optimization, and a multipurpose theme that must work on a plethora of devices.

The WordPress professional should know the plugin and theme market – and know enough about WordPress to be able to tweak the code, when the client needs something unique.

### Sandra's advice:

- Theme: Perth was used in her final dissertation. This theme was chosen because it resembled her sketches for the page layout.
- Plugin: Page builder was used in order to create responsive pages.

### Daniel's Advice

- Familiarise yourself with one multipurpose theme.
- WooCommerce is vital for webshops.
- Obtain payment and shipping knowledge.
- Suggestions towards a WordPress certification.

# The History of WordPress

---

This chapter will give you an outline of the development of WordPress. The source code that later became WordPress dates back to 2001. On the web most sites are deprecated after two years, or so. WordPress has been around since 2003.

The CERN scientist Sir Tim Berners Lee invented the World Wide Web in 1989 ("[The Birth of the Web](#)"). Therefore, when WordPress was born the internet was still in its expansive phase. It had existed for little over a decade. The roots of WordPress and the vision for the system is defined in the WordPress Codex:

"WordPress was born out of a desire for an elegant, well-architected personal publishing system built on PHP and MySQL and licensed under the GPL. It is the official successor of b2/cafelog. WordPress is fresh software, but its roots and development go back to 2001. It is a mature and stable product. We hope by focusing on web standards and user experience we can create a tool different from anything else out there." [Codex: About WordPress](#)

WordPress began in 2003 when *Matt Mullenweg* and *Mike Little* forked the open source project [B2 Cafelog](#). *B2 Cafelog* was an open source blogging system that dated back to 2001.

In 2004, the concept of plugins was introduced. Here is a definition of plugins:

"A WordPress Plugin is a program, or a set of one or more functions, written in the PHP scripting language, that adds a specific set of features or services to the WordPress site, which can be seamlessly integrated with the site using access points and methods provided by the WordPress Plugin Application Program Interface (API)."

[Codex: Writing a Plugin](#)

A year later in 2005, *themes* were introduced. Themes are the *skins* of WordPress pages. The architecture separates *content* from *presentation*. The content is stuff like articles, images or videos. The presentation is the layout, behaviour, and general look of the web page.

From 2010 new demo themes were introduced each year. That's why I will follow the development of themes from 2010 to the present day.

The first theme was Twenty Ten, named after the year it appeared. All themes sum up the state of the WordPress art. Often the themes relate to web development paradigms, such as *responsive web design*. Seen in this way the WordPress themes sum up important design and tech trends on the web. That's why knowing the history of themes is important.

Now the recent theme is Twenty Seventeen, launched in December 2016. A separate chapter is devoted to the development of this theme by the open source community behind WordPress.

In the following chapter, you will find the history of the WordPress themes.



# The Core Themes

The theme is one of the most important concepts of WordPress. The general idea behind a content management system is a strict separation of:

- *The Content Layer* and
- *The Presentation Layer*.

*The Content Layer* consist of menus, information structure, articles and multimedia productions, like video, infographics and photos.

In Content Management, *The Presentation Layer* could be compared to layout or templates, as we know it from publications. The "theme" is the presentation layer in WordPress terminology. Therefore, the theme defines what things look like for the visitors on a web page and when an author publishes her work on the webpage.

Since the introduction of themes in 2005, a new theme has been introduced every year. In many ways you can say that the themes sum up the ambitions of WordPress. The Twenty X themes display the WordPress state-of-the-art.



Michael Heilemann designed the first theme "Kubrick"



The first theme was presented when WordPress 1.5 "Strayhorn" was released on February 17th 2005. The founder of WordPress, Matt Mullenweg, introduced the theming concept:

"In 1.5 we have created an incredibly flexible theme system that adapts to you rather than expecting you adapt to it. You can have your entire weblog run through a single file, just like before, or you can literally have a different template for every single different category. It's as much or as little as you want."

**Source: WordPress News.**

The new theme and its possibilities was called "[Kubrick](#)" and designed by Michael Heilemann:

"Of course we wanted to showcase the new flexibility with a new theme that took full advantage of it and was aesthetically pleasing to boot, so the new default theme for WordPress is the beautiful Kubrick by Michael Heilemann." (Source WordPress News op.cit.)

To the present day reader, Heilemann's theme may appear to be rough. Nevertheless, the theme became extremely popular. There's a blue header and blog posts beneath it. To the right, there is a sidebar with a search function, pages, new posts, archive and more. On the other hand, many themes still follow that basic design (e.g. wordpress.org (2015) and wordpress.org (2016)).

Heilemann tells the story of the design in his blog:

"By some stroke of fortune this open source theme developed for WordPress 1.2 may be the most widely used template ever designed, having been used by millions of sites and ported to over thirty completely different platforms.

"Automattic made it the default template with WordPress 1.5 in 2006, a position it kept until 2010 when it was finally retired." <http://binarybonsai.com/kubrick/>

The pages import header and footer content via `get_header()` and `get_footer()`. A PHP loop will fetch content from the MySQL database.

In addition, the "Kubrick loop" is very similar to [the loops](#) used in WordPress themes today, see [index.php l. 11 - 39](#) (Github).

The "Kubrick Loop" was the first theme. Now, a new core WordPress theme is introduced every year. A website owner may choose to design and code her own theme. Nevertheless, the WordPress core themes are the state-of-the-art.

I have chosen to focus on the most recent themes dating from 2010 to 2017. Many code snippets are still used in the WordPress themes. The web pages are divided into a "body" (e.g. `index.php`).



# Themes: 2010 – 2017

---

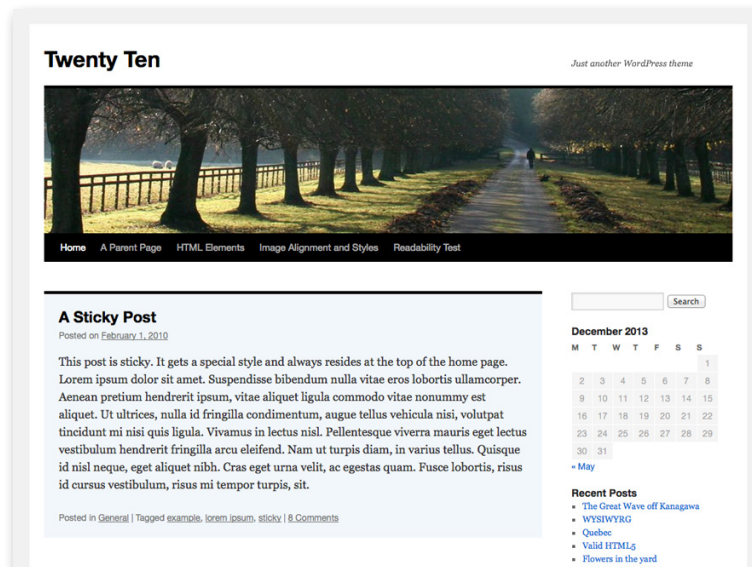
In this chapter you can follow the development of the core themes in WordPress from 2010 - 2017. In this period, the themes were named "Twenty" followed by the year of introduction of the theme to the WordPress-core.

If you know where to look, the history of each theme is saved among the theme files. WordPress themes are distributed via online servers. You can download the themes as zip-files, and install them. The core themes are available via the Dashboard.

Among all these files, we find a file called **readme.txt**. In the readme file, you will find a short description of the theme. If new WordPress features are implemented, the intended target group for the theme, and much more.

In the theme folder, you can find a screenshot of the intended design, as it would look in a web browser. The file is called **screenshot.png**, which is a very obvious name for the file.

Here are the themes from 2010 and on. In the quotes from the readme-files, I have marked important ideas, and concepts with a **bold** font.



Twenty Ten

## Twenty Ten

"The 2010 theme for WordPress is stylish, customizable, simple, and readable -- **make it yours** with a **custom menu, header image, and background**. Twenty Ten supports **six widgetized areas** (two in the sidebar, four in the footer) and **featured images** (thumbnails for gallery posts and custom header images for posts and pages). It includes stylesheets for print and the admin Visual Editor, special styles for posts in the "Asides"

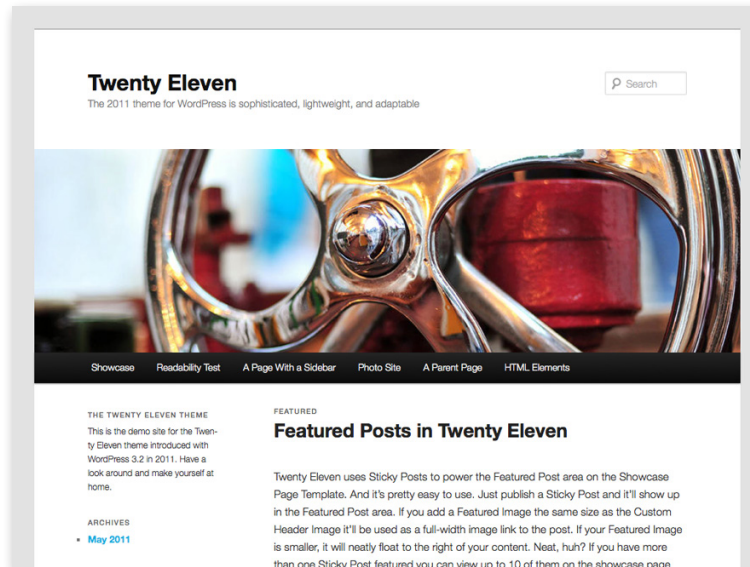
and "Gallery" categories, and has an optional one-column page template that removes the sidebar." (wordpress.org 2010)

To sum up the quote from the readme.txt file:

- Options to personalize menus, header image.
- In the right sidebar, some *widgets* are displayed: a calendar and recent posts.

In fact, most of the basic theme concepts apply to all themes dating from 2010 and onwards.

What happened on the computing stage in the year 2010? Apple introduced the iPad to the market that year. During this year, Samsung's [Galaxy Tab](#) series also began. The tabs and smartphones came to revolutionize the ways web pages were used. This development led to new design paradigms. Web sites should look good on screens and mobile devices. This development had a huge impact on WordPress in the years to come.



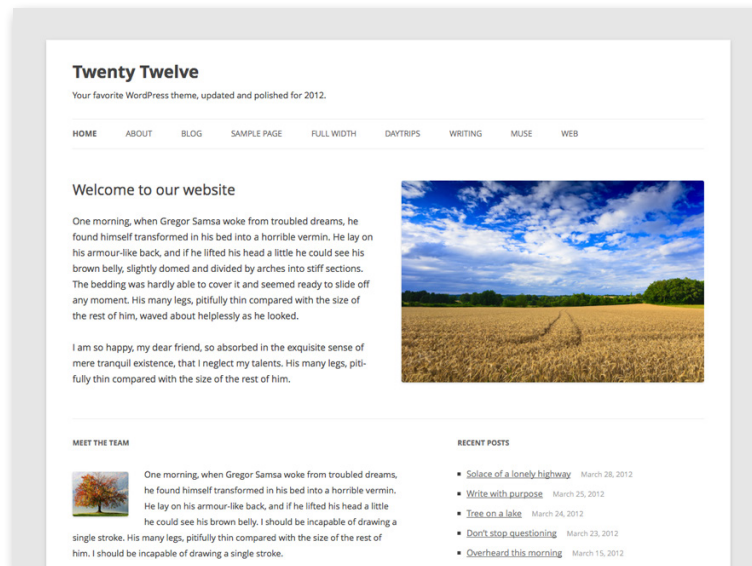
*Twenty Eleven*

## Twenty Eleven

"The 2011 theme for WordPress is sophisticated, lightweight, and adaptable. **Make it yours with a custom menu, header image, and background** -- then go further with available **theme options for light or dark colour** scheme, custom link colours, and three layout choices. Twenty Eleven comes equipped with a **Showcase page template** that transforms your front page into a showcase to show off your best content, widget

support galore (sidebar, three footer areas, and a Showcase page widget area), and a custom **"Ephemeria" widget** to display your Aside, Link, Quote, or Status posts. Included are styles for print and for the admin editor, support for featured images (as custom header images on posts and pages and as large images on featured "sticky" posts), and special styles for six different post formats." (wordpress.org 2011)

Twenty Eleven also focussed on customisation. Now the user was able to manipulate the header image, and the background of the web page. Posts and pages could have special header images. Twenty Eleven had styles for print too, but the description does not mention anything that targeted mobile devices.



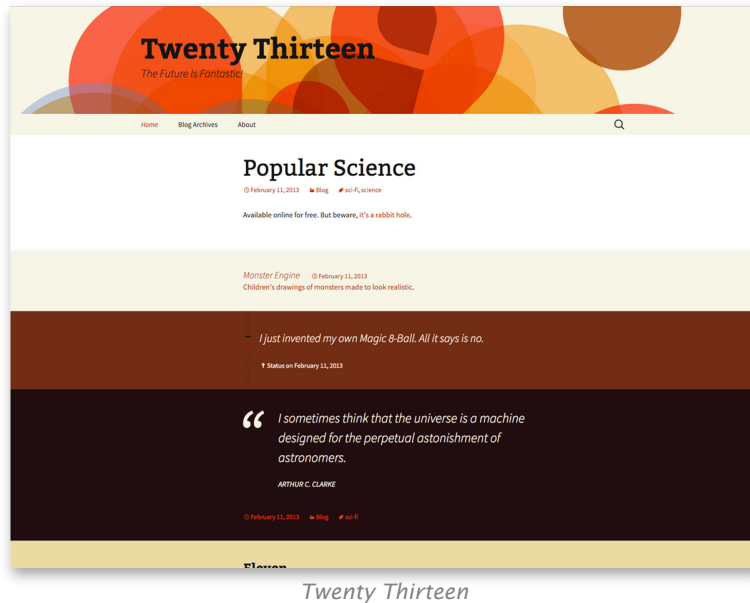
Twenty Twelve

## Twenty Twelve

"The 2012 theme for WordPress is a **fully responsive theme** that looks great on **any device**. Features include a front-page template with its own widgets, an optional display font, styling for post formats on both index and single views, and an optional no-sidebar page template. **Make it yours with a custom menu, header image, and background.**" (wordpress.org 2012)

In 2012 the web developers claimed that the theme was designed for "... any device". Here we begin to see the impact of *tablets* and *smartphones*. On the web, more and more people entered the web pages from mobile devices. Responsive web design was the answer, or rather *one content for many devices*.

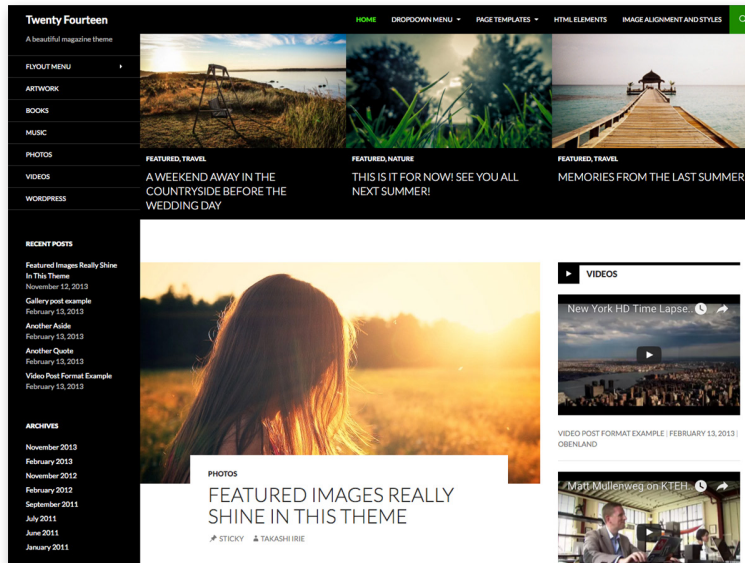




## Twenty Thirteen

"The 2013 theme for WordPress takes us **back to the blog**, featuring a full range of post formats, each displayed beautifully in their own unique way. Design details abound, starting with a vibrant colour scheme and matching header images, beautiful typography and icons, and a **flexible layout that looks great on any device**, big or small." (wordpress.org 2013)

Twenty Thirteen seems like a back to the roots design. WordPress originated from blogging, hence returning to a blog design is natural. Again, the designers had to make a solution "... that looks great on any device".



*Twenty Fourteen*

## Twenty Fourteen

"In 2014, our default theme lets you create a **responsive magazine website** with a sleek, modern design. Feature your favourite homepage content in either a **grid or a slider**. Use the three widget areas to customize your website, and change your content's layout with a full-width page template and a contributor page to show off your authors. Creating a magazine website with WordPress has never been easier." (word-

press.org 2014)

The Twenty Fourteen theme targeted magazines. Several ezines and online papers used WordPress, among them according to a 2014 blog post: The New Yorker, BBC America, and MTV News. [Source WP Beginner August 10th 2014.](#)

In 2014 *Responsive Web Design* became something of a megatrend among web developers and gurus. The web designer Etan Marcotte coined the concept

of responsive web design in 2010. In the ezine *A List Apart* he published the article "Responsive Web Design" (and used the word "responsive" 57 times!):

"Rather than tailoring disconnected designs to each of an ever-increasing number of web devices, we can treat them as facets of the same experience. We can design for an optimal viewing experience, but embed standards-based technologies into our designs to make them not only more flexible, but also more adaptive to the media that renders them. In short, we need to practice *responsive web design*." (Marcotte 2010)

By 2014 responsive web design was the thing to do. The trend is obvious in titles such as:

- "Responsive Performance With WordPress" (Dusablon 2014).
- "Responsive Design with WordPress" (Casabona 2014).
- "Responsive Design Workflow" (Hay 2013).
- "The Responsive Web Design Handbook" (Carney 2014).

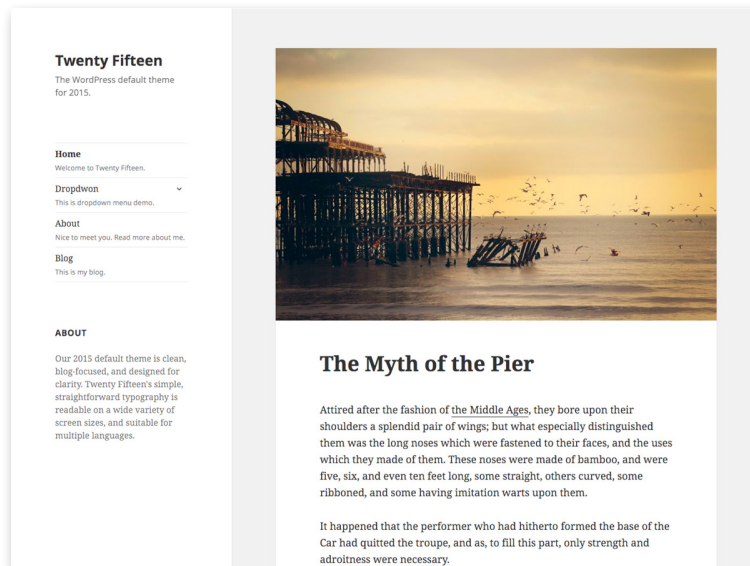
The responsive designers envisioned a web where:

- Content should be produced in one place and distributed to several media channels.
- The web designers used scalable grids in order to target "many devices".
- Web developers struggled with "fluid images" that would scale to relevant sizes.

WordPress embraced the responsive web design mega-trend, and gave it tangible form in the core theme.



Rather than tailoring disconnected designs to each of an ever-increasing number of web devices, we can treat them as facets of the same experience. We can design for an optimal viewing experience, but embed standards-based technologies into our designs to make them not only more flexible, but also more adaptive to the media that renders them. In short, we need to practice responsive web design. – Marcotte 2010

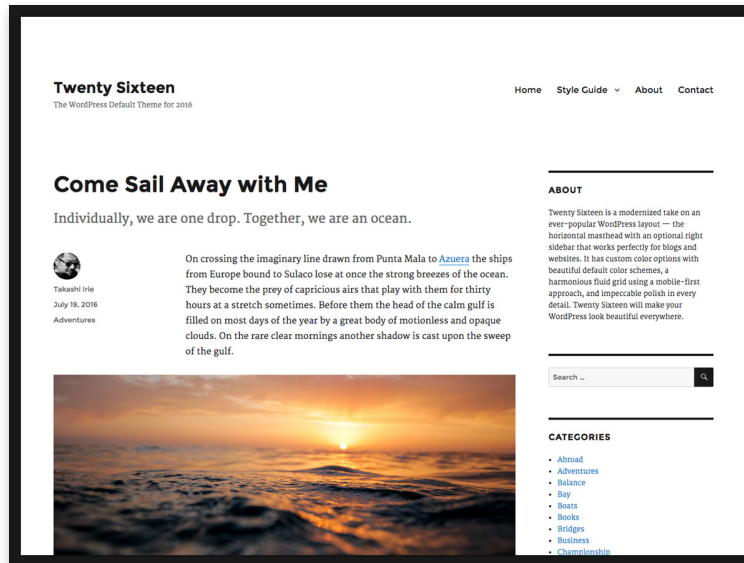


*Twenty Fifteen*

## Twenty Fifteen

"Our 2015 default theme is clean, **blog-focused**, and designed for **clarity**. Twenty Fifteen's simple, straightforward typography is readable on a **wide variety of screen sizes**, and suitable for **multiple languages**. We designed it using a **mobile-first approach**, meaning your content takes centre-stage, regardless of whether your visitors arrive by **smartphone, tablet, laptop, or desktop computer**." (wordpress.org 2015)

By 2015, the pendulum swung back to the blog design again. The web pages still had to adapt to several devices. Now the WordPress developers embraced the mobile first concept. It is a design process where the developers begin with a focus on small devices, like mobile phones. Gradually the designer would develop solutions for larger screens, such as the MacBook Pro's Retina Screen.



Twenty Sixteen

## Twenty Sixteen

"Twenty Sixteen is a modernized take on an ever-popular WordPress layout — **the horizontal masthead with an optional right sidebar** that works perfectly for **blogs and websites**. It has **custom colour options** with beautiful default colour schemes, **a harmonious fluid grid** using a **mobile-first approach**, and impeccable polish in every detail. Twenty Sixteen will make your WordPress look beautiful everywhe-

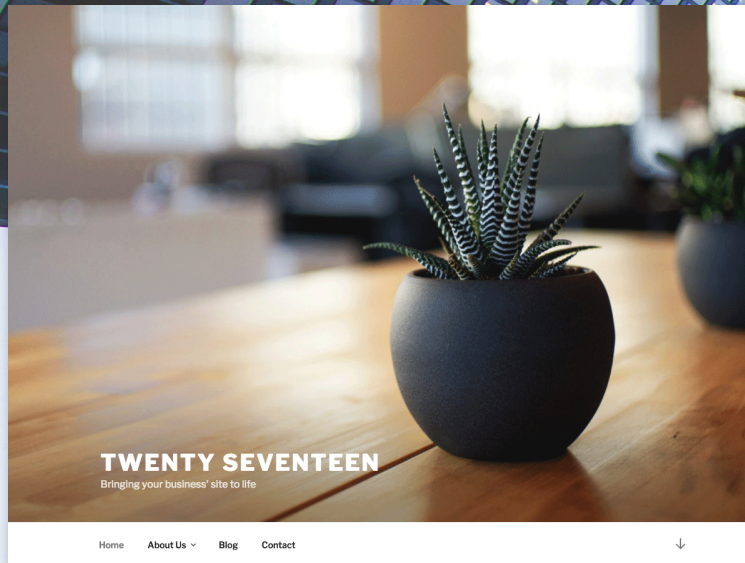
re." (wordpress.org 2016)

Both Twenty Fifteen and Twenty Sixteen mark yet another return to the blog and the webpage roots. The themes are *responsive*, but now Responsive Web Design is more or less a standard. It is something you are expected to do.

The design process is *Mobile first*. It's a design method where the designers begin with design for small devices. The following design iterations tar-

get bigger screens, say tablets - and end out with the design big screens (Carney 2014: 26 pp., 49 pp.)

The *mobile first* movement realized, that you couldn't just downscale the big screen design to a small screen. Each device has special needs and priorities (Hay 2013).



*Twenty Seventeen*

## Twenty Seventeen

"Twenty Seventeen brings your site to life with header video and immersive featured images. With a focus on business sites, it features multiple sections on the front page as well as widgets, navigation and social menus, a logo, and more."

The development of Twenty Seventeen began during the fall of 2016. The development of the theme happened at the same time as the research for this book.

I decided to follow the development of the theme from the early stages on Github to the integration of Twenty Seventeen in the core-WordPress code. The features of the theme are summed up in this statement:

"Twenty Seventeen brings your site to life with **header video and immersive featured images**. With a **focus on business sites**, it features **multiple sections on the front page** as well as widgets, navigation and social menus, a logo, and

more. **Personalize** its **asymmetrical grid** with a custom colour scheme and display your multimedia content with post formats. Our default theme for 2017 works great in many languages, for any abilities, and on any device." (wordpress.org 2017)

The Twenty Seventeen theme "focuses on business sites". The big image style is used in order to create an *immersive* feeling. As an alternative to the big image, a video could be launched.





In fact, the big image / video is the header part of a one-pager. Here you could say that the header grew almost to full screen. In 2017, most web browsers and networks can manage videos. The videos can be streamed from YouTube.

Doing so is very easy. The user can stream a video from YouTube by pasting the video's URL in the "Custom Settings" found in the Dashboard.

WordPress targets businesses by the big image and multimedia design megatrend. Here WordPress makes a statement:

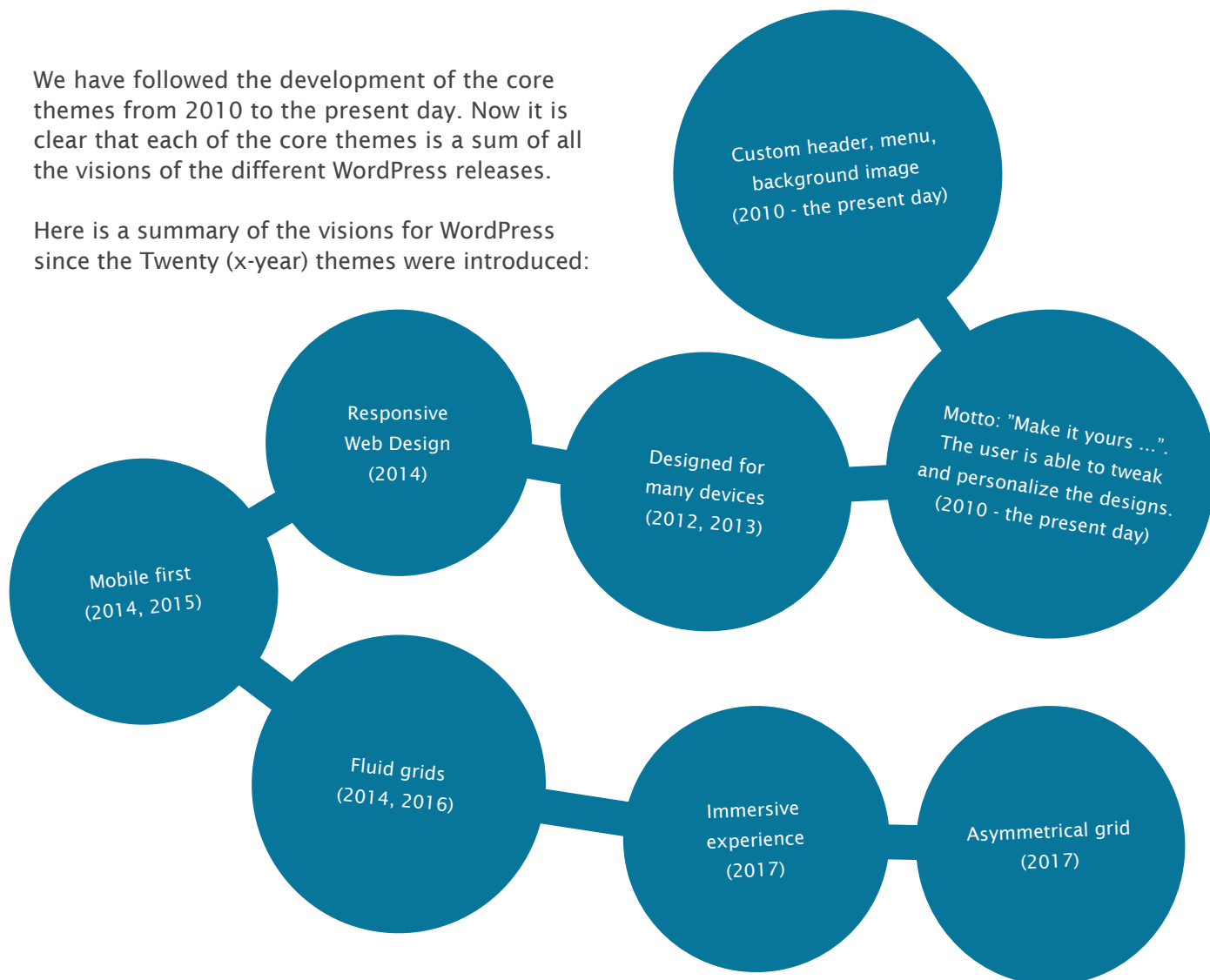
WordPress transformed from a blogging tool to a powerful multimedia communication channel ready for business.

# Themes as Visions for WordPress

---

We have followed the development of the core themes from 2010 to the present day. Now it is clear that each of the core themes is a sum of all the visions of the different WordPress releases.

Here is a summary of the visions for WordPress since the Twenty (x-year) themes were introduced:



The seven themes named after the year they were made highlight many web development megatrends, or paradigms. Most of the themes are made for bloggers. This is natural since WordPress started as a fork from a blogging system.

A few themes target very specific segments, such as magazines (2014) or business sites (2017) Twenty Seventeen's target group is business professionals. In spite of these efforts on behalf of WordPress I guess that in general WordPress is perceived more as a blogging tool than a Content Management System.

The new theme Twenty Seventeen is designed for business with immersive design and state-of-the-art multimedia integration. The WordPress themes from the period 2015 to 2017 all reflect the ambitions and visions of the community. The themes respond to the technical development. When tablets and other mobile devices became popular, the themes changed which was reflected in the design megatrends, like Responsive Web Design, and Mobile First.

In many ways, the development of the WordPress themes reflects the history of the internet – and the devices we use on the web.



# Anthropology and Open Source

---

How is WordPress made? In this chapter, I will follow the development of the latest version of WordPress, and focus on the creation of the core theme.

Each year a new WordPress theme is made. We have seen how the themes sum up most of the creative ambitions and visions of the open source community behind WordPress.

You can see the themes as a demonstration of what WordPress is able to do now. Often themes target well-defined user groups. In this chapter, I will follow the development of the most recent WordPress theme (WordPress.org 2016; wordpress.org 2017; Hou-Sandi 2016)

In order to follow the development of a WordPress theme I decided to use a field study. It is a research method inspired by anthropology. I became part of the community, and followed the development closely. During the field study, I participated in a series of online communities where WordPress is made.

In classical anthropology, the researcher could travel to a tribe far, far away. The anthropologist would live among the natives, and log observations on a daily basis about the society. So was the research method of Margaret Mead (1901 - 1978). In 1928 she travelled to the distant island *Ta'u* in Samoa. Here she studied the culture, and focussed on adolescent girls (Mead 1928).

The method used in this dissertation is inspired by Schein's et al's philosophy of anthropology as expressed in "Organizational Culture and Leadership" Schein (2010).

Here Schein focus on three organizational layers:

- [Artefacts and behaviours](#)
- [Espoused values](#)
- [Basic assumptions](#)

# Artifacts

The *artefacts* layer consists of visual or organizational structures and processes.

”Artifacts include the visible product of the group, such as the architecture of its physical environment; its language; its technology and products; its artistic creations; its style, as embodied in clothing, manners of address, and emotional displays; its myths and stories told about the organization; its published lists of values; and its observable rituals and ceremonies.” (Schein 2010 / quote: 2010 4th. ed. p. 25)

Schein’s philosophy is relevant in connection with observations of the WordPress development process. One could ask how can you make observations of a community that does not meet on a geographical location, like a studio? How can you observe hierarchies, power structure, group dynamics, or social

The following table presents the artefacts of WordPress:

Artifacts	Note
Language	English
Architecture	Digital culture
Technology	Code: MySQL, PHP, HTML, CSS, Javascript
Artistic Creations	Themes, the logo, beauty of code
Style	Business style vs. Blogger style
Myths, stories	Myth: Free Open Source Software
Published values	<a href="#">Philosophy of WP</a> , Freedoms of WP
Rituals, Ceremonies	Announcements on WP news blogs

interaction in online fora? I have realised that this is possible.

Therefore, the ”architecture” for the WordPress community is not a building of block. It is rather a ”digital culture” (Creeber and Martin, 2009) with participants

from all over the world. With Schein we could ask:

- How are decisions made in the online WordPress culture?
- How are the projects managed?
- Who’s in charge?



You could argue, that Schein's model reveals the more or less hidden forces governing WordPress. In a vast international organisation someone has to make a decision. If we follow the development of a theme, and look at the decisions, the hierarchy of influence becomes visible.

In "Personal Connections in the Digital Age" Nancy Baym points to a vital challenge:

"People can't take for granted that the people with whom they have relationships share their attitudes. Figuring out when and how to use media to communicate with each other is part of figuring out what it means to relate well to others." (Baym 2010 op.cit. p. 149)

The quote shows that WordPress is developed in a very complex architecture of media. Figuring out where to suggest something, or how modify code can be a daunting task for a new contributor in the open source community.

According to Edgar Schein, the *product* is an artefact. The end product is the blogging / Content Management System WordPress. The sole purpose for the community is to develop the product. Therefore, the community has a common goal.

In order to study this community, I had to create a several profiles on Slack, Github, and [make.wordpress.org](https://make.wordpress.org). Then I could follow and participate in the proceedings of the work in the online community, and work with anthropological observations.

# Espoused Values

Schein defines espoused values as declared strategies, goals, or philosophies in an organisation.

"Such beliefs and values often become embodied in an ideology or organizational philosophy, which then serves as a guide to dealing with the uncertainty of intrinsically uncontrollable or difficult events." (Schein 2010: 27)

Sometimes there is a gap between the espoused values and the behaviour of the organization:  
"... In many organizations espoused values that reflect the desired behaviour but are not reflected in observed behaviour (Argyris and Schön, 1978, 1996). For example, a company's ideology may say that it values people and that it has high quality standards for its products, but its actual record in that regard may contradict what it says." (Schein 2010: 27)

Espoused Values	In the WordPress organisation
Ideology	Code should be open and free
Philosophy	Everyone can participate in making a better product. WordPress is a collective effort
Handle Conflicts	WordPress is not entirely democratic. There is a hierarchy of developers, and designers. The lead profiles have the power to make decisions in case of conflicts.

Consequently, the researcher should be aware of the difference between what:

- The company says it does.
- What the company actually does.

According to Schein, it is important to:

"... Discriminate carefully among those that are congruent with the underlying assumptions that guide performance, those that are part of the

ideology or philosophy of the organization, and those that are rationalizations or only aspirations for the future." (Schein 2010: 27)

The philosophy of WordPress is found in the online publication [Our Philosophies](#). The first part gives the espoused value of the WordPress software from the point of view of the end user: "Great software should work with little configuration and setup. WordPress is designed to get you up and running and



fully functional in no longer than five minutes.” (op.cit.)

The software is developed with the 1% rule in mind:

”The number of people who create content on the internet represents approximately 1% (or less) of the people actually viewing that content” ([”1 % Rule – Internet Culture”](#))

The people who give feedback to WordPress may be a minority. So the general rule for the developers is to listen to the users, for instance at Word-Camps, or utterances about the product in online fora.

Online fora are among the espoused values. [The official communication channels](#) are:

- WordPress Trac
- WordPress SVN and Git
- Slack
- WordPress Blog
- Development Blog
- Mailing Lists

The quality of the code is also an important espoused value:

”WordPress is a big project with thousands of contributors. It’s important to follow best practices so that the codebase is consistent and readable. And furthermore that changes are easy to find and read, no mat-

ter if the code is five days or five years old.” ([Handbook: Best Practises](#))

WordPress has well defined ideas about the poetry of code. Why connect poetry and code? Perhaps because coding can be said to be a creative art form.

The handbook: Best Practises links to several standards for the code, e.g. HTML, JavaScript, PHP and CSS. These documents tend to be detailed to the extreme (there is a Latin word for it:”Ad Nauseam”). A typical example is this:

### According to WordPress, this is correct CSS:

Here the selectors have one line each. Since the lines are shorter, the code should be easier to read:

```
#selector-1,  
#selector-2,  
#selector-3 {  
    background: #fff;  
    colour: #000;  
}
```

### According to WordPress, this code sucks:

Why? Because, in this case, the selectors are listed in one long line. Longer lines are harder to read:

```
#selector-1, #selector-2, #selector-3 {  
    background: #fff;  
    colour: #000;  
}
```

```
#selector-1 { background: #fff; colour: #000; }
```

The authors state that “tabs, not spaces”, must make the indentations. Each selector “should be on its own line”. In this way, WordPress attempt to create code that’s easier to read. Read more about the WordPress coding standards [here](#).

Coding standards are important among *espoused values* in the WordPress community. It is important, that WordPress students learn to write not only code that will “get the job done”. They should learn to follow *coding style manuals*. Writing code that looks professional is an important skill in the craft of code. If they can follow the standards of WordPress, they could quickly adapt to other coding standards.

Both code samples would work just fine in most browsers. In this case, WordPress simply prefers a certain code style, and the programmers are expected to follow such guidelines.

Important espoused values in the WordPress organization are these:

- Contribute with Code
- Contribute with Testing
- Best Practices
- Tutorials

You could argue that it can be very hard to find what you are looking for, for instance in the monstrous wiki called the [Co-dex](#). It is easy to get lost in the haze of information.

## Basic Assumptions

Schein defines *basic assumptions* as almost unconscious *a priori* assumptions, thoughts and feelings. Often it is hard to define the basic assumptions, because it is akin to tacit knowledge. Schein defines the basic assumptions like this:

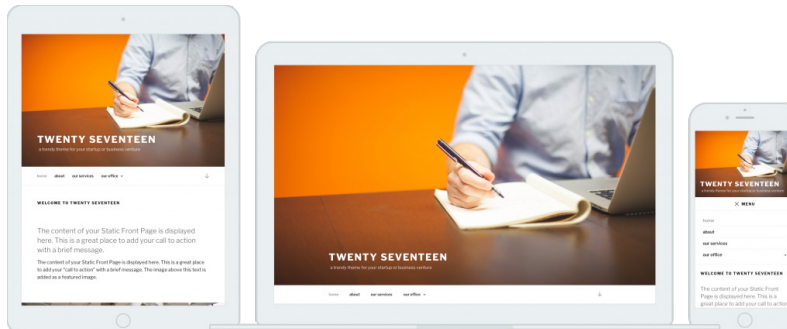
"When a solution to a problem works repeatedly, it comes to be *taken for granted*. What was once a hypothesis, supported only by a hunch or a value, gradually comes to be treated as a reality." (Schein, 2010 op.cit.)

The almost unconscious nature of the basic assumptions will manifest as soon as someone violates them:

"In fact, if a basic assumption comes to be strongly held in a group, members will find behaviour based on any other premise inconceivable." (Schein, 2010 op.cit.)

According to Schein, the basic assumptions surface in conflict situations. People from all over the planet make WordPress.

Conflicts are bound to come now and then. How does the WordPress community cope with such conflicts? Here we have to look at the WordPress hierarchy, or power structure.



*Twenty Seventeen Promo. Design: Mel Choyce et al.*







For WordPress developers, it is interesting to follow the management of open source projects. Who is in charge? How are deadlines or milestones defined? Schein has a point in saying that conflicts are important cues towards understanding the hierarchy in an organisation.

The anthropological part of the book is based on my observations of the proceedings in the WordPress core-theme developer groups during the development of the Twenty Seventeen theme from august 2016 to January 2017. The groups communicated and exchanged code and designs via Github and Slack.

In the section "About WordPress" / Credits (4.7), the names of all developers are displayed. The lead developers (2016 / 2017) are:

- 1. Matt Mullenweg, Automattic, Cofounder, Project Lead
- 2. Dion Hulse, Brisbane, Australia
- 3. Andrew Ozz, Vancouver, BC
- 4. Helen Hou-Sandi, Jersey City, NJ
- 5. Mark Jaquith, Washington, D.C.
- 6. Andrew Nacin, Brandon, FL, USA

In the core-theme group Helen Hou-Sandi, David Kennedy (Theminator @Automattic) and Mel Choice (Product Designer at Automattic) played a prominent role during the development of WordPress during the fall of 2017. As lead developers, they had the power to decide *Yay* or *Nay* on whatever suggestions the community came up with.

 <div><b>Matt Mullenweg</b> Release Lead</div>	 <div><b>Andrew Nacin</b> Lead Developer</div>	 <div><b>Mark Jaquith</b> Lead Developer</div>
 <div><b>Andrew Ozz</b> Lead Developer</div>	 <div><b>Helen Hou-Sandi</b> Lead Developer</div>	 <div><b>Dion Hulse</b> Lead Developer</div>

*Some WordPress lead profiles. Screen dump from "About WordPress"*

The lead role of Automattic is evident. In a blog post evaluating the WordPress 4.7 development process Jeffrey Paul stated:

"Whoever is leading the release has final decision. That's why they're a lead. That much should be clear." (Jeffrey 2016)

The open source community behind WordPress is ruled neither by anarchy nor by democracy. The *espoused value* is "one ring to rule them all" (Tolkien 1954). In the end Automattic - and especially Matt Mullenweg have the power to decide.

During the development process deadlines, bug scrubs and alike were initiatives by Automattic.

In order to pass into the core, a new theme must meet the standards of any WordPress theme. The theme-reviewers are important figures here. David Kennedy is a key-figure. His bio on the profile pages, quotes:

"I work as a Theminator (that's my official title) at the ever-awesome Automattic, the company behind WordPress.com and many other fine web products. I wrangle themes for WordPress.com, making them the best they can be and ensuring everyone can find a theme they love. I'm an accessibility evangelist who loves the open web and open source code. I contribute to WordPress and the WordPress community." <https://profiles.wordpress.org/davidakennedy>

As soon as Twenty Seventeen reached a certain maturity, David Kennedy was the one to give the ritual announcement to the WordPress community. September 9th, 2016 Helen Hou-Sandi published a ritual blog post on WordPress.org:

## Say Hello to Twenty Seventeen

"It's that time again: time to build a new default theme for WordPress! WordPress 4.7 will launch with a brand new theme – Twenty Seventeen. Designed by Mel Choyce (@melchoyce), Twenty Seventeen sports a modern look and will make a good base for any business website or product showcase."

(Hou-Sandi, 2016)

Who are these lead characters: Helen and Mel? According to <https://profiles.wordpress.org/helen> Helen Hou-Sandi lives in Jersey City, NJ.

"I'm the Director of Platform Experience at 10up and a WordPress Lead Developer."

On LinkedIn, we find more details. Apart from working at Automattic, Helen is a musician. In the WordPress fora, Helen was the one who defined deadlines, made agendas, and often she was responsible for summaries after meetings.

Helen introduced yet another lead profile – enter the designer:

"Mel [Choice] will keep an eye on all things design during the creation of Twenty Seventeen. Laurel Fulford (@laurelfulford) and David Kennedy (@davidakennedy) will assist her, leading

the theme's development. Lots of opportunities exist this year for getting involved with Twenty Seventeen – we need your help, and lots of it!" ([make.wordpress.org](https://make.wordpress.org))

The lead profiles are clearly defined. The influence of Automattic is evident, since the lead designer, David Kennedy and Laurel Fulford are employees at Automattic.

The rules of code syntax were also defined:

"Twenty Seventeen will also use plain CSS — it won't use pre-processors in the development of the theme. This will keep it simple, making the theme easier for everyone to understand, quicker for anyone to modify and better to maintain in the long run." (op.cit.)

The code should be "easy ... to understand". During the development, certain coders wanted to speed up the process by pre-processors and similar. The theme code is a didactic tool for WordPress. The suggestions for automation were just ignored. In such cases, we see the influence of Automattic.

The Automattic employee Mel Choice designed Twenty Seventeen. On the [WordPress profile pages, Mel Choice's](#) bio gives this information:

"Boston-based UI/UX designer, WordPress core contributor, craft beer fan, and unrepentant sci-fi/fantasy geek. I'm a Product Designer at @Automattic."

Seeing Automattic employees in the core themes group is hardly a surprise. Automattic is *the* most important stakeholder here.



## Beginning development on Github

---

Twenty Seventeen was based on an unpublished theme. The theme features a big image or video on the front page. As we have seen the intended target group is business sites. In a short statement from September 10th 2016 the Automattic employee Laurel Fulford [@laurelfulford](https://twitter.com/laurelfulford) told, that Twenty Seventeen is based on the unpublished theme Lodestar: "Twenty Seventeen is a fork of a yet-to-be-launched theme, *Lodestar*. Right now, the styles in GitHub reflect that original theme. The styles need to be updated to match Twenty Seventeen's design here:

[https://cloudup.com/cR\\_df4x-feeG](https://cloudup.com/cR_df4x-feeG)" [See the statement on Github](#)

The developers used *Github* as their mutual platform for version control during the initial stages development. Github was initiated by Linus Torvalds and the Linux developers in 2005 [Source: "10 Years of Git: An Interview with Git Creator Linus Torvalds", Linux.com](#). In the interview, Github is defined:

"(... in 2005 ...) Linus Torvalds, the creator of Linux, took the challenge into his own hands and disappeared over the week-end to emerge the following week with Git. Today Git is used for thousands of projects and has ushered in a new level of social coding among programmers." (Cloer 2015)

Github started as a versioning tool for Linux developers. Github's platform for collective software development proved valuable for many developer communities. WordPress is among them. During the development of Twenty Seventeen, around 60 developers participated on Github. They could raise issues, and suggest solutions to the issues. They would share code snippets. In addition, the developers who owned the Github repository could integrate the suggested changes in the code.

- Link to: [Issues on the Github Twenty Seventeen repository](#)

A typical sample of a code contribution began 19th October 2016. The Swiss developer @Grappler suggested a solution to an issue. Actually @Grappler is the nick of Ulrich Pogson. The code issue was the following: in Twenty Seventeen, there was no support for the *Urdu* language and fonts. To the post he added [this suggestion for Urdu support](#).

@Melchoyce tried to implement the code. She could not see the expected results in the "Site Language List". Here @Grappler responded that the *Urdu* translation of WordPress was "... only at 95% ..." which explained the missing language. Then Mel Choice accepted the suggested code:

"@melchoyce merged commit 7362081 into WordPress:master"

Or, in human readable form:

The Automattic employee Mel Choice accepted the suggested code from the developer. In this way, Automattic listens to the open source community. Developing Twenty Seventeen on Github seemed like a very public affair. Nevertheless, there is no doubt that Automattic is in charge.

This is but one typical code contribution among the 235 small and large contributions to the Twenty Seventeen theme from the WordPress community on Github. Counting the comments to the issues gives an impression of the most debated important contributions to the theme:

- "Better support for non-Latin font fall-backs" (Labels: design, enhancement, help wanted, revisit later)
- "Explore Custom Colors" (Labels: design, enhancement)
- "Back to top arrow?" (Labels: ally, enhancement, PR needed)
- "SVG in place of icon font" (Labels: enhancement, has PR)
- "Captioned images with no alignment get misaligned" (Label: bug)

[For a complete list, follow this link.](#)

The top five most debated issues give an impression of the topics. Most of the issues among them had labels such as enhancement or design. A WordPress theme is much more than code. The lead designer is an important character.

Sorting by the label "has priority" suggest, what the developers really wanted to do with the design. Here the top five list is:

1. "SVG in place if icon font"
2. "Speed up all the animations"
3. "Featured image sizes / size limits"
4. "Linked image in footer widgets has underline"
5. "Update Screenshots.png"

[The complete list is here.](#)

It is hardly a surprise, that design is the most important issue when the topic is a theme. Multimedia content is important. Animations, videos and graphic file formats are important. For instance, the developers wanted to work with an open source icon theme. The community suggested using SVG (i.e. Scalable Vector Graphics, an xml-based vector file format).

In SEO, the speed of a web page is important. This may explain the collective efforts to "speed up animations" and work on sizing featured images. The observations based on the collective work on the Twenty Seventeen theme on Github lead to the following conclusions:

- Automattic defines initiatives, such as deadlines and milestones.
- Automattic makes important decisions.
- The community contribute with code suggestions for enhancements and ideas.

After developing Twenty Seventeen on Github, the code was refined on *Slack*. Here the party is more "private WordPress party" than the very open Github development site. There were weekly meetings, and so-called *bug-scrubs* where errors were corrected. In principle the proceedings were similar.

On Slack, the developers are able to:

- Suggest enhancements
- Contribute to code
- Conduct tests
- Deploy code

On October 18th 2016 "Theminator" David A. Kennedy made this ritual announcement: "The team behind Twenty Seventeen has reached that point that we're ready to propose Twenty Seventeen as the new default theme for WordPress in 4.7. An ambitious theme focuses on a creative home page and an easy site setup experience for users.

Like last year, with Twenty Sixteen, the development process happened on GitHub. The theme will be merged into WordPress from the beta period on, and development will move to Trac. Some remaining tickets will move over to Trac at that time.” [Kennedy on make.wordpress.org](http://kennedyonmake.wordpress.org)

On the WordPress news-pages, such announcements mark important occasions. David Kennedy is the self-proclaimed “Theminator” of WordPress. The title suggests the lead role of David Kennedy when it comes to core-themes.

The contributions from the community were:

- Triaging issues.
- Providing code reviews.
- Testing and recommending language specific font stacks.
- Improving the theme’s accessibility.
- Browser and device testing.
- Numerous bug fixes, code tidying and countless improvements.

Kennedy suggests that the developers should test the theme:

”While contributors have tested Twenty Seventeen on various devices and browsers throughout the development process, edge cases still exist. Please test Twenty Seventeen in as many different environments as you can.”

There’s a certain ritualistic element to such announcements. They mark that a certain development level has been reached.

On Github, we cannot only follow the code. We can also see the debate and thoughts that led to the code. From an anthropological point of view, it is interesting to follow the proceedings of the open source community. The next chapter will focus on anthropology.





# Conclusion: Schein, Organization and Development

---

With Schein (2010) we can analyse the open source part of WordPress as an organization. WordPress has its own artefacts. There is also espoused values and even basic assumptions. All of this became visible during my anthropological study of the development of the standard theme Twenty Seventeen.

## Artifacts

There are many artefacts in the WordPress world. If you go to a WordCamp, there are t-shirts, badges, keyrings, pens, WordPress logos etc. all over the place.

On webpages, you often see a footer message like "Powered by WordPress".

According to Schein (2010), the product is an artefact. Therefore, a WordPress blog or content management system is an artefact.

The product WordPress is not developed in a building, factory, or alike. The various online fora (Slack, Trac, and Github) are artefacts too.

## Espoused Values

WordPress is open source. The software is free for anyone to use. They can modify the content to whatever use they want. Be it commercial or not.

In the online manuals, WordPress has the espoused value: "listen to the users and developers". With Schein (2010), one could ask:

"... But do they listen to their peers?"

Here the answer is: yes, they do. Nevertheless, at the end of the day the final decision about code and design is made by the lead profiles. They are either hired or chosen by the company behind WordPress: *Automattic*.

## Basic Assumptions

The basic assumptions in open source communities is that code must be free for anyone to use and improve. In this way, Open Source leads to better code. This philosophy derives from the GNU-philosophy defined by Richard Stallman et al.

"Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things."

[GPLv3, url: https://www.gnu.org/licenses/gpl.txt](https://www.gnu.org/licenses/gpl.txt)

Patents are perceived as "evil" or rather a threat, as you can see in the following quote:

"Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free." [GPLv3 op.cit.]

The GPLv3 license does not define how the software should be developed. In practise, WordPress is developed as the collective effort of a very large international community, with *Automattic* as the lead. The organization resembles other open source communities lead by a *Benevolent Dictator for Life* (BDFL). Linus Thorvalds is

the BDFL of the operative system *Linux*. Tim Berners-Lee BDFL of the W3, the consortium behind the *World Wide Web* standards. Guido van Rossum of the programming language *Python*.

Matt Mullenweg with *Automattic* is the equivalent of a BDFL in the WordPress hierarchy of power and influence.

Open source societies are ruled neither by democracy, nor by crude anarchy. At the end of the day, the open source projects have a very strict *power hierarchy*. In several ways, this observation is in conflict with the *storytelling* of open source.

A hierarchy of power is evident. Progress comes, because *Automattic* and Matt Mullenweg define the deadlines, agendas, accepts or rejects. That being said, the WordPress society is very open, even kind, towards new ideas. Here they follow the

ideas from Raymond (1997):

"Release early. Release often. And listen to your customers."

In the first part of "WordPress in the Classroom" we have followed the open source community during the creation of a WordPress layout. Open source is not a thing that falls down from the sky. Real people, like the designer Mel Choice, make WordPress.

WordPress is a living organisation, and the anthropological point of view led to a deeper understanding of WordPress artefacts, espoused values and the basic assumptions.

Part two of this series is "Diving Into Themes". Here you will learn what a theme is, and how to create custom WordPress themes – as a tweak of an existing theme or how you can create a theme from scratch.



# Part two:

# Diving Into Themes

The second part focuses on WordPress frontend development. You will learn how to create your own design - either from scratch, or tweak an existing theme by a child theme. Part two will also introduce some new trends in WordPress frontend development, such as Nodejs features and command line tools.



# The Theme-code

---

So, let us dive deep into the theme code. First you will get a general introduction to WordPress themes. I assume that you have a general knowledge about HTML, CSS and perhaps some JavaScript. If you know some PHP it's fine. However, you certainly don't have to be an expert in PHP in order to follow the instructions. The code samples from the book are available via Github:

- As simple as it gets - a three file theme: [petj-mini-theme](#)
- A minimum viable theme: [petj-mvp](#).
- A Bootstrap scaffold theme: [bootstrap-f16-skeleton](#)
- A Nodejs sample (features: Bower, Gulp, Sass): [nuitt](#)

The purpose of the themes petj-mvp and petj-mini-theme is to create a meta-theme. That theme demonstrates *how themes work*.

The following themes are a bit more complicated. The Bootstrap scaffold theme is an implementation of a WordPress theme based on Twitter's Bootstrap. If you know how to create a static Bootstrap website, you can take it to the next level and add content with WordPress as the content manager.

The last theme will give you a little glimpse of the endless possibilities with Nodejs and rapid web development. We'll plunge into technologies like SASS stylesheet pre-processing, and some automation with Gulp. When the SASS-files changes, Gulp will automatically create a style.css for WordPress.

Nevertheless, before we dive into the deep, let's have a look at the basic building blocks of any WordPress theme.

First, you'll have to install WordPress. If you own a web hotel, and a bit of luck, there might be a "WordPress one click install" option. Go ahead and create one!

If not you can install a local-host server on your computer. [XAMPP](#) or [MAMP](#) are popular solutions for Windows and Mac users. Linux should install a LAMP server, click the link, and follow [these](#) instructions.

As soon as your server is up and running, you will be able to download and install WordPress.

- [Download WordPress](#). It is a zip file. Unzip the files to a folder shared by your server.
- Follow these instructions in order to [install WordPress](#) on your server.

You will be able to access WordPress by your browser as soon as it is installed.

# Theme Building Blocks

The professional WordPress developer should understand the basic building blocks of any theme. First we must figure out where the files are in a typical WordPress installation. The content layer and the presentation layer are separated in WordPress. That much you already know. A frontend developer will create a presentation layer for WordPress. A WordPress theme is the presentation layer.

```
<?php if (have_posts()) : while (have_posts()) : the_post(); ?>
  <h2>
    <a href="<?php the_permalink(); ?>"
      rel="bookmark" title="Permanent Link to
      <?php the_title_attribute(); ?>"
      <?php the_title(); ?>
    </a>
  </h2>

  <div class="entry">
    <?php the_content(); ?>
  </div>

  <?php endwhile; else : ?>
    <p><?php _e('Sorry, no posts matched your criteria.');
  <?php endif; ?>
```

*A theme is like a puzzle made by a mixture of ordinary HTML, and WordPress specific PHP snippets. The loop is a good example. Content is added from the database and presented on the web page.*

So, where are the files we need? In a typical WordPress installation, you find the theme files in:

```
.. pathToWordPress/wp-config/themes/yourTheme
```

It is good practice to save the theme in a folder with the same name as the theme. You can expect to find Twenty Seventeen in this folder:

```
.. pathToWordPress/wp-config/themes/twentyseventeen
```

## Three Important Files

---

In web design, the stylesheet defines the overall layout, typography, graphic backgrounds and colours. In standard HTML or PHP the name of the stylesheet does not matter. That's not the case in a WordPress theme. Here the primary stylesheet must have the name **style.css**. Of course you may add other stylesheets if need be. However, style.css is the place where WordPress will look for the name of the theme, the author, copyright info and much more. If style.css does not exist, WordPress will assume either that the theme is broken or that it does not exist.

It is possible to make a WordPress theme with as little as two files. In practice, however, three files are much better. Try to create a folder with files:

- **index.php** (will generally display the main content)
- **style.css** (here are the styles and the theme name is defined in a comment)
- **functions.php** (in this file we add scripts, and other functionality)

You can try out a theme like this from my Github repository: [petj-mini-theme](#). Let us have a closer look on the most important files in any WordPress theme.

# The style.css

The first file we will look at is the **style.css**. It is a cascading style sheet. Stylesheets define the presentation layer. They may even create the UX look and feel with film like animations and transitions. In a WordPress theme, a stylesheet in the theme's root folder must have the name **style.css**.

The **style.css** is an ordinary CSS file, as you know it from other web pages. However, WordPress expects the following comment in the first part of the CSS (see illustration above).

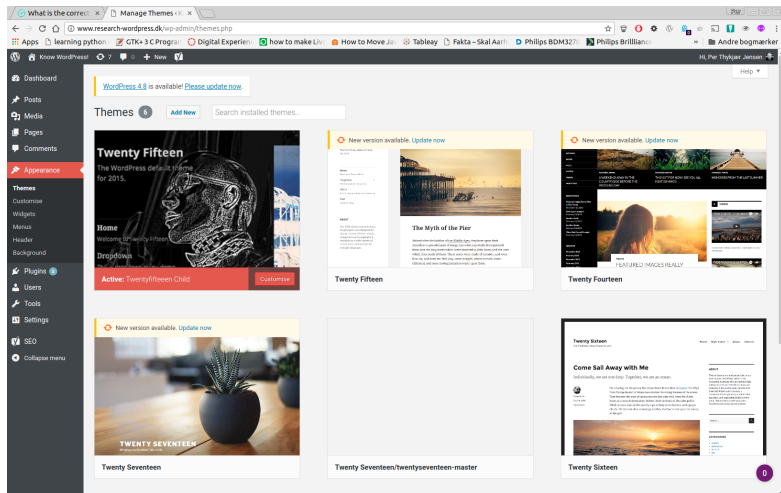
When **style.css** is present in the root folder of your theme WordPress will know that the theme exists. Actually, you can see it on the Dashboard's overview of the theme.

```
/*
Theme Name: Petj Mini Theme
Theme URI: http://github.com/asathoor/petj-mini-theme
Author: Per Thykjaer Jensen
Author URI: http://www.multimusen.dk
Description: The most minimalistic theme I've ever made. Just
for demonstration.
Version: 1.0
License: GNU General Public License v3
License URI: http://www.gnu.org/licenses/gpl-3.0.html
Text Domain: petj-mini-theme
*/

/* add your styles here */

body { ... styles ... }
```

*The comment section in style.css*



*If you add an image called **screenshot.png** in the theme folder, you will be able to see the image over the theme's name on the Dashboard.*

Now try to add a screenshot of your design, and save the file as **screenshot.png**. If you look on the Dashboard / Appearance you will get an image of the design on the Dashboard's theme section. Exactly as you can see it from core themes like Twenty Fifteen. The screenshot.png is optional, but it's nice to get an impression of the themes on the Dashboard. You don't have to use a screenshot since you can add a logo or any arbitrary picture that you fancy.

By now, WordPress is able to see your design on the Dashboard. However, in order to present content in a browser we need some mark-up.





## The index.php

---

On most web pages, the index file is the root of the file hierarchy. In `index.php`, you write the markup used by the stylesheet and scripts. Here you have to decide on a markup that will present the content in a nice way. Therefore, the combination of the index and the style file are the key elements that define the "presentation layer".

WordPress will read the content from the MySQL tables in the database. WordPress will read the content by the so-called "WordPress Loop". Many template tags, or PHP objects, are available for the developer. But for clarity, it is a good idea to start with a very basic loop.

Create an **index.php** file with content along these lines:

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

<?php
// The Loop
// Make a better loop - @url: http://codex.wordpress.org/The_
Loop
if (have_posts()) {
    while (have_posts()) {
        the_posts();
        // add template tags see @url: http://codex.wordpress.org/
        Template_Tags
        // Here are some template tags:
        the_title('<h3>', '</h3>'); // write the title in h3 tags
        echo '<article class="loopArticle">';
        the_content(); // here's the content
        echo "<article>";
    }
}
?>
<?php wp_footer(); ?>
</body>
</html>
```

Here you see a very primitive loop. The loop starts like this:

```
if (have_posts()) {  
    while (have_posts()) {
```

If there are posts or pages. Now you can add the markup for the content presentation along with the so-called "template tags". The template tags will get the content from the database. In the markup, you see sample template tags, like

- `the_title()` - will present the title of pages and posts.
- `the_content()` - will present the content, or body text, of the posts along with images, videos and similar content.

The developer can design quite elaborate loops with dates, author names, author images ([gravatars](#)), and much more. In the online WordPress Codex, you'll find [a long list of template tags](#) for your markup.

Working with just one index file like this will soon prove impractical. Most web pages need more than one layout to fit all content. Often we use different designs for landing pages, front-pages, video pages, categories and so on.

That is why we use partial files. PHP will glue the design together when we include partials, such as headers, navigation bars, and footers.

There is a file naming convention. In **header.php**, we write the mark-up for the top of most pages, and possibly some header navigation. In **footer.php** you can save the footer information like copyright notes, contact information and similar.

Saving the loop in a separate file is a good idea. In the code below the loop is saved in `loop.php`. The file is included in the `get_template_part("loop")` function.

As soon as **header.php**, **sidebar.php** and **footer.php** are ready, you can include them in your markup.

The files are glued together via WordPress template tags, such as `get_header()`. The tag does exactly what it is called. `get_header()` will import the **header.php** file, and the file **loop.php** will populate the web page with content from the database.

```
<?php
/**
 * file: index.php
 */
get_header(); // import header.php

// THE LOOP
get_template_part("loop"); // import loop.php
get_sidebar(); // import sidebar.php
get_footer(); // import footer.php
?>
```

*From the file **index.php***

## header.php and footer.php

---

The **header.php** will contain all the HTML header information. In the head section, you normally link to your stylesheets. But in WordPress, the best practice is to use the *enqueue* in **functions.php**. That's why there's no link to the style.css in the sample code. WordPress is also shipped with the JavaScript library jQuery. But in a theme, you have to enable jQuery. This may seem a bit cumbersome, but some themes just do not need jQuery.

Now you may note, that this header lacks most of the important SEO tags. You could of course add the SEO tags manually to **header.php**, or use some built in features from `bloginfo()`, as shown in the `<meta charset="<?php bloginfo('charset')"; ?>" />`.

A simple **header.php** might look like this:

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

    <!-- enable jQuery -->
    <?php wp_enqueue_script("jQuery"); ?>

    <!-- load: title, scripts, css etc. -->
    <?php wp_head(); ?>
</head>
<body <?php body_class(); ?>>
```

Perhaps you wonder: where is the title tag? From SEO theory, we know that the title tag is a *very* important information for the search engines, and the browser want the tag too. However, the title tag is not used in this version of header.php. Or at least so it seems.

Why? Actually, the title tag is defined in the file **functions.php** (See next chapter). WordPress will get the information about the site for the title tag from the settings on the Dashboard. Or rather: from the MySQL database.

In order to get a properly formatted title tag add these lines to your functions.php:

The template tags `get_footer()` will load the file **footer.php**. Moreover, `get_sidebar()` will load the file **sidebar.php**. In `footer.php`, `wp_footer()` must be added before the body ends. `wp_footer()` will load footer scripts. The scripts are defined in **functions.php**.

Here is the content of a very simple **footer.php**:

The `footer.php` could of course be more elaborate than this. Try to add some typical footer mark-up, like copyright information, phone numbers and alike above the `wp_footer()` function.

```
/**
 * file: functions.php
 * purpose: Title Tag Support
 */

function theme_slug_setup() {
    add_theme_support('title-tag');
}
add_action('after_setup_theme', 'theme_slug_setup');
```

```
<!-- file: footer.php -->

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

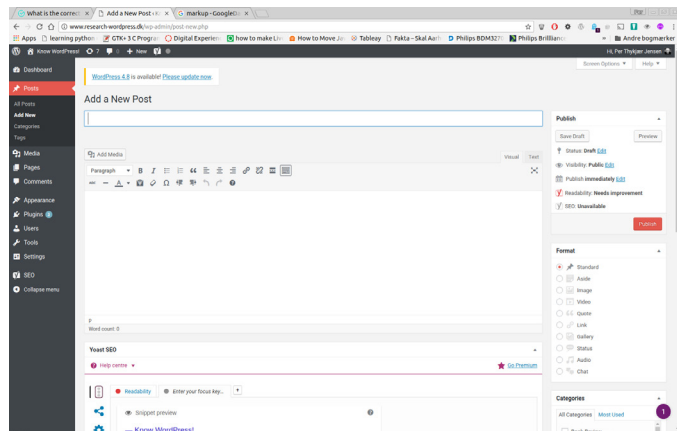
# functions.php

**Functions.php** is used in order to add extra functionality and options to a WordPress theme. If a function is declared in this file, it is globally available on all pages in the theme. However, the changes and additions in `functions.php` will only be available for the particular theme:

"The functions file behaves like a WordPress Plugin, adding features and functionality to a WordPress site. You can use it to call functions, both PHP and built-in WordPress, and to define your own functions. You can produce the same results by adding code to a WordPress Plugin or through the WordPress Theme functions file."  
[From: Functions File Explained](#)

The **functions.php** can [add theme support](#) for numerous features, add stylesheets, scripts, and even modify the way the Tiny MCE Editor behaves (That is the editor that is used on the Dashboard, for instance when you write a new post or page) and [much more](#).

**Functions.php** can add content by so-called "[hooks](#)" in the template. For instance, stylesheets will be added to the header via the hook `wp_header()`. That's why it is important to add the `wp_header()` towards the end of the `<head>` section in the **header.php** mark-up.



*The Tiny MCE editor on the Dashboard. Here it is used to add a new post. You can add functionality to the editor in the file `functions.php`.*

```
// from: header.php
// Twenty Seventeen

<?php wp_head(); // "hook" ?>
</head>

<body <?php body_class(); ?>>
```

The stylesheet is also added through the hook `wp_head()`. Again, **functions.php** does the trick. Stylesheets are enqueued like this:

JavaScripts and additional stylesheets are added after a similar manner. Here are links to a few samples of additional functionality via `functions.php`:

- [Add your own mark-up: improved SEO sample.](#)
- [Add extra buttons to the text editor Tiny MCE.](#)
- [Remove bloated header code.](#)
- [Support for title tag](#)

```
function my_enqueue_style() {
    wp_enqueue_style('core',
        get_template_directory_uri()
        . '/style.css',
        false);
}
add_action('wp_enqueue_scripts',
    'my_enqueue_style');
```

Have a look at the "Support for title tag" code sample from the link above. Here you can see how the title tag is added, so there is no need to write one in your **header.php**. Remove the bloated code from the header that will result in improved performance, and as a consequence better SEO ranking. The "WordPress mark-up" is not that different from ordinary HTML.

The major differences are:

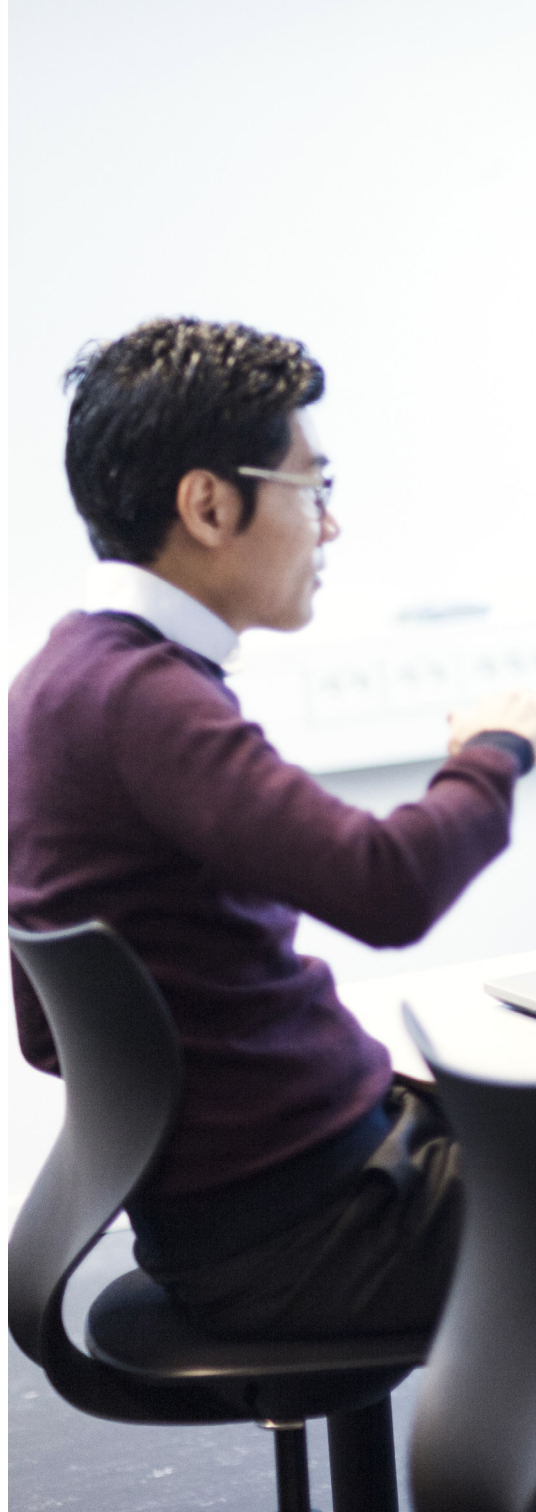
- The title tag should be activated in `functions.php`. From a SEO point of view it is good practise, because WordPress will format a title tag with the site name, and the name of the landing page.
- Scripts and stylesheets should be enqueued in `functions.php`.



If you really want to, it is of course possible to hardcode the links to scripts and stylesheets as you would do it in HTML in your **header.php**. But sometimes it's nice to be able to remove unwanted scripts. If the header is hardcoded that's not possible. You can remove an unwanted script, and even add extra scripts on custom pages:

```
<?php wp_dequeue_script($script); ?>
```

The best practise is to add the scripts and stylesheets via the enqueue functions.





# Menus

---

In most WordPress themes, the administrator can create menus on the Dashboard. Often you can choose to display a menu at several locations in the design. Menus are used for navigation. They are important UX cues, since they define the user's options and choices during her interactions with the web page.

In a theme, you could choose to hardcode a menu. But if the information architect must consult a web developer every time he fancies a menu change, it will become expensive for the owner of the website.

In a WordPress theme the developer can define "menu areas" *anywhere* in the mark-up. The information architect can create several menus on the Dashboard. Each menu must then be assigned to a "menu area".

WordPress will loop out the menus as unordered lists <ul>. As you can see, there are front-end and backend issues here. Here's a guide to add a menu area.

1. The theme must support menu locations. The menu location names are defined in functions.php.
2. The menu must be located in relevant spots, such as header.php or sidebar.php ( the positions for the menu depends on the design ).
3. On the Dashboard, the end user must create a menu.
4. On the Dashboard, the end user shall assign the menu to a menu location.
5. The menu's style and behaviours must be defined. The developer must use her skills to style the mark-up via stylesheets or scripts.

## Ad 1) Menu Location Names ( functions.php )

First, the menu name must be defined:

You can define several menus in the same function. If you need two or more menus, then just add them to the function.

## Ad 2) Add the Menu locations in the Markup

As soon as the menu is defined in functions.php, it is possible to define the location, wherever you want the menu to appear in the mark-up. For instance you could place a menu in **sidebar.php**:

WordPress will return the menu as a list. The list should be styled via CSS or perhaps JavaScript. This list should be styled via CSS or perhaps JavaScript. jQuery is part of any WordPress installation since the Dashboard uses it.

```
function register_my_menu() {
    register_nav_menu('sidebar-meu',
        __('sidebar Menu'));
}
add_action('init', 'register_m_meu');
```

*From functions.php*

```
<!-- menu location: sidebar menu -->
<?php wp_nav_menu( array(
    'theme_location' => 'sidebar-menu')
); ?>
```

*From sidebar.php*

```
<div class="menu">
    <ul>
        <li class="page_item page-item-703">
            <a href="http://localhost/wordpress/blog/">
                Blog</a></li>
        <li class="page_item page-item-701">
            <a href="http://localhost/wordpress/front-page/">
                Front Page</a></li>
        <!-- etc. -->
    </ul>
</div>
```

*Sample markup generated by PHP*

Remember to enable jQuery in the **header.php**. The script below is jQuery, so:

```
<?php wp_enqueue_script("jQuery"); ?>
<?php wp_head(); ?>
</head>
```

*jQuery enabled*

As the menu is used on all pages, **footer.php** is the logical place to add a jQuery script that will give some dropdown functionality:

- The jQuery menu code is inspired [this tutorial](#).

In jQuery a `.toggle` will hide something if it's visible and display something if it's hidden. In this case the CSS class `.children` is hidden. So we cannot see the "children" in the menu.

`$( '.page_item_has_children' ).hover` is activated when the mouse hover over a menu item. Then the `.slideDown()` and `.slideUp()` will animate the menu and it will look like a dropdown effect.

```
<script>
( function($) {
    $('.children').toggle();

    $('.page_item_has_children').hover(
        function(){
            $(this).children('.children').slideDown(200);
        },
        function(){
            $(this).children('.children').slideUp(200);
        }
    );
} )(jQuery); // jQuery end
</script>
```

*A jQuery script sample*

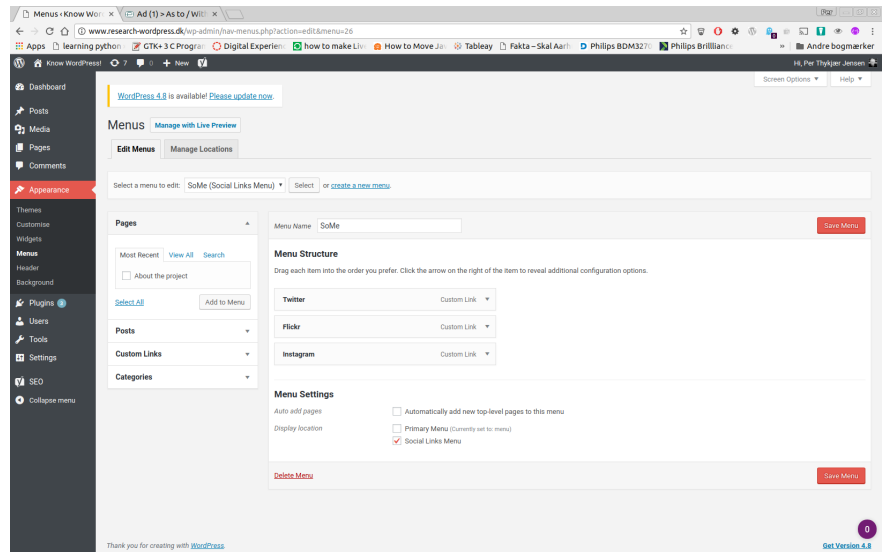
Now we have some mark-up, styles and JavaScript for a simple dropdown menu. The menu prepared the menu in functions. php. jQuery is enabled and used by the menu. The next step is activating the menu on the Dashboard.

## Setting up the menu on the Dashboard

When the menu areas are ready in the code, you can assign menus to the theme's different menu-locations via the Dashboard. Head for the menu section under appearance.

- Create a menu on the Dashboard. Save it.
- Add the menu to the theme location. Save it.

If all's well the menu should be visible on the theme page. Normally the menu will loop out as an unordered list. Then you can style the list in your CSS or by JavaScript.



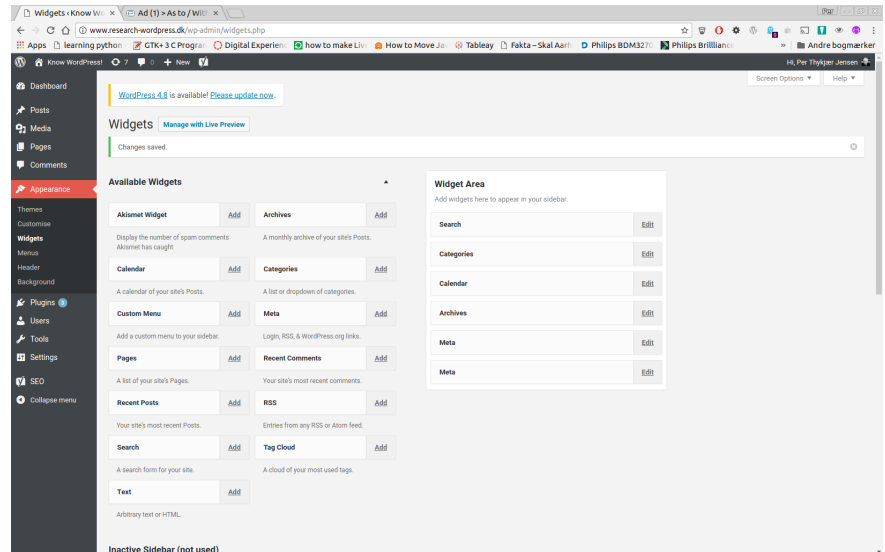
*A menu on the Dashboard.*

# Theme Widgetizing

Widgets are small add-ons for your web site, like calendars, meta-information, RSS-feeds, and similar. WordPress define widgets, as:

"WordPress Widgets add content and features to your Sidebars. Examples are the default widgets that come with WordPress; for Categories, Tag cloud, Search, etc. Plugins will often add their own widgets." [Codex: "WordPress Widgets"](#) (Visited: 20170301)

In the quote above it is assumed that a widget must be placed in a sidebar. Nevertheless, widget areas are not confined to the sidebars at all. Widget areas may be defined *anywhere* you want widgets in the mark-up.



*Widgets on the Dashboard. Left: a list of available widgets. Right: Widgets used in the Widget Area.*



From a semantic point of view widgets are *asides* in HTML5. So, they may or may not be placed in a sidebar. The sidebar.php is just one of the standard WordPress files. The sidebar is included like this:

```
<aside id="sidebar">
  <?php get_sidebar(); ?>
</aside>
```

*get\_sidebar() will fetch sidebar.php*

Like menus, the widget areas are prepared in **functions.php**, and invoked somewhere in the mark-up. [Here is a functions.php](#) widget area sample:

```
**
* Two Widget areas
Header det bottom eller bottom fx bottom sidebar...
*/
function petjmv_widgets_init() {
    //first widget area
    register_sidebar( array(
        'name'          => 'Home right sidebar',
        'id'            => 'home_right_1',
        'before_widget' => '<aside>',
        'after_widget'  => '</aside>',
        'before_title'  => '<h3>',
        'after_title'   => '</h3>'
    ) );

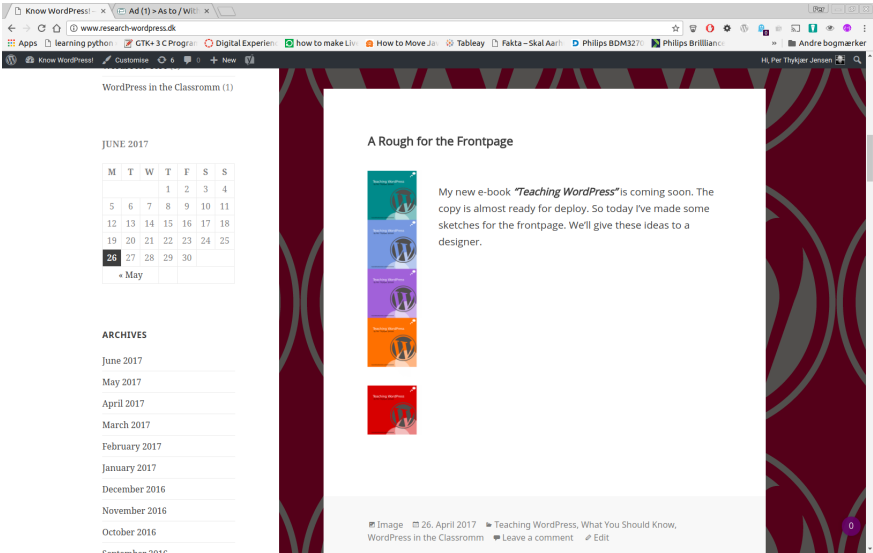
    // second widget area
    register_sidebar( array(
        'name'          => 'Bottom Sidebar',
        'id'            => 'bottom_sidebar',
        'before_widget' => '<aside>',
        'after_widget'  => '</aside>',
        'before_title'  => '<h3>',
        'after_title'   => '</h3>'
    ) );
}
add_action('widget_init', 'petjmv_widgets_init');
```

*Widget areas defined in functions.php*

The next step is to place the widget area(s) somewhere in the mark-up, e.g. in [sidebar.php](#):

Now the widget areas are defined in functions.php. The dynamic\_sidebar( 'name' ) snippets are added in relevant lay-out position. Then the user is able to activate the widgets on the defined areas of the theme.

```
<!-- file: sidebar.php -->
<div id="sidebar">
  <!-- snippet: add a sidebar -->
  <?php dynamic_sidebar('home_right_1'); ?>
</div>
```



*Here you see the widgets as they appear on a web page.  
From a child theme of Twenty Fifteen.*

## Advanced Loops: Snippets and Tags

---

A typical WordPress page is very similar to a newspaper page. There's a header, a content section and a footer. Most WordPress pages have asides. Here you see widgets, such as calendars and similar meta information. All parts are glued together with WordPress-specific PHP snippets, the so-called template tags.

Here is a typical WordPress template:

```
<?php
/**
 * File: page-NAME.php
 */
get_header(); ?>

<main>

    <?php
    /**
     * A very simple Loop
     */
    if (have_posts()) {
        while (have_posts()) {
            the_posts();
            //
            echo '<article>';

            the_title('<h3>', '</h3><div class="content">');
            the_content();
            echo "</div>";

            echo '</article>';
            //
        } // end while
    } // end if

</main>

<aside>
    <?php get_sidebar(); ?>
</aside>

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

The PHP snippets `get_header()`, `get_footer()`, and `get_sidebar()` will load content from the files **header.php**, **footer.php** and **sidebar.php**. These files contain code that will be used on all pages. Basically, custom pages follow the same structure as the **index.php**.

The post or page *content* is loaded from the *MySQL* database via "The Loop". The loop in my example is very primitive. A more elaborate loop would link to individual pages and each of them could have their own specially crafted loops, depending on what the developer wants to display. For a good example, please refer to the [Loop Examples](#) on the Codex.

The list of available template tags is very long. There is no point in going over all of them here. Here is the link to a page that will link to most of the important template tags:

- [The template tags](#)

Here are two loop samples from [my template demos](#) on Github:

- [A simple loop](#)
- [A more complex loop](#)

The loop is a combination of WordPress specific PHP snippets and ordinary HTML. Here is a sample loop, where the title will link to the page:

```
<!-- formatting the title -->
<h2>
  <a href="<?php the_permalink(); ?>"
    rel="bookmark"
    title="Permanent Link to
      <?php the_title_attribute(); ?>"
    <?php the_title(); ?>
  </a>
</h2>
```

- `The_permalink()` will echo the link to a page or post.
- `the_title_attribute()` will give a short text explaining the page.
- `the_title` is the actual page title from the document.

The title is stored in the MySQL table *wp\_posts*. So `the_title()` will echo content of the column *post\_title* from the database. Of course the developer could write her own PHP with SQL reads from the database. `the_title()` is a PHP function that does the same thing. It is easy to use such template tags. Knowing such development shortcuts is a great advantage for a web developer.

If the template contains a post. php the developer could give the post a design with a custom-made mark-up. WordPress developers can develop custom design to *any* page.

# The Template Hierarchy

---

In WordPress, you can create webpages with a tweaked design for certain pages. If you want a particular design for pages in the video category you can even create special templates for individual pages and posts.

In order to create a custom page you must name the files in a certain way. Here are some examples:

- `frontpage.php` - will be used as the front page.
- Templates for: categories, videos, etc.
- Even a particular post can have its customized design.

But how do we do this in practice? The [template hierarchy](#) is the key to custom-made pages. A quick and dirty template hierarchy example is this:

1. Create a static *page* and give it the title "My Portfolio" in the dashboard.
2. Publish the page.
3. Click "Screen Options" and make "Slug" active.
4. Find the name of the slug somewhere next to the page editor.

Normally, the slug is a short version of the title. Let's just say that the page name is "My Portfolio", and that the slug is "my-portfolio". Now you can create a custom-made page. If you consult the [template hierarchy info graphic](#) you can see that your portfolio page should be named `page-my-portfolio.php`. Here is the mark-up for the page:

```
<?php
/**
 * Template: page-my-portfolio.php
 */
get_header();
?>

<!-- html from here -->
<h1>Hello World</h1>

<?php

get_footer(); ?>
```

You could add any creative mark-up or JavaScript between the `<?>` and the `<?php>`. If you don't want to display the header or footer you could drop the `get_header()` and `get_footer()` too, and add whatever alternative markup you need. Therefore, you could add a PHP gallery, fancy animations, full screen

maps – and much more.

In a similar way, it is possible to create custom pages for individual posts, pages, categories, etc.

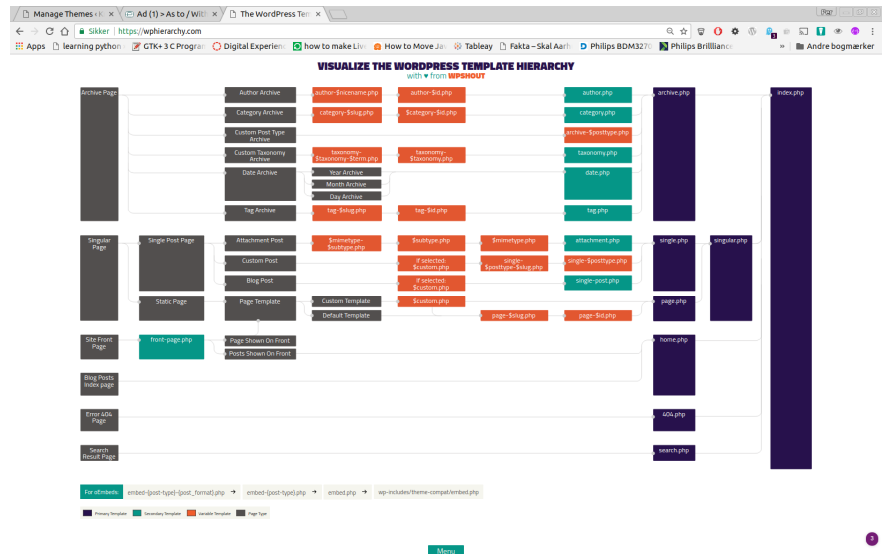
Now, enter the Dashboard and view the page that you just wrote. The result should be a page

saying "Hello World". That's it. In order to see the actual content a loop is necessary.

**Tip:** If you have access to the MySQL database, the slug is found in the table `wp_posts`. Here the slug is found in the column `post_name`.

In the template hierarchy you can see, that a page must be named after this convention:

- `page-SLUG.php`
- Hence the filename is:
- `page-my-portfolio.php`



The *template hierarchy info-graphic*.



## Partial Conclusion: The Theme Building Blocks

---

Now we have seen the major building blocks of a WordPress theme. We've learned to:

- Create modular pages, split in header.php, footer.php and content.
- Seen that **functions.php** is the place where you add functionality. And that you can add functions for a particular theme here.
- You can create any number of custom pages: for categories, pages, posts and more.

Whether your theme is made from scratch, or the theme is an elaboration on an existing theme, it is necessary to have knowledge about:

- The theme files.
- WordPress file naming conventions.
- The WordPress Loop.
- When to use a theme and whether a child theme is necessary.

Now you know the basic files in any WordPress theme. If you need more information, the "[Theme Handbook](#)" is a very good resource.



# Why you need a child theme

---

In the classroom - and certainly in real business life - time matters. In some cases, you simply don't have enough time to create a full theme from scratch. Luckily, there are thousands of themes out there. Why not use a ready-made open source theme as a template and just tweak the theme's code here and there?

Well, it is possible to do so. But if the publishers update the original theme, your tweaks can vanish like smoke. The solution to this problem is *create a child theme*.

In the classroom, the child theme has an advantage. The code of the child theme is separated from the "parent theme". The child theme code is saved in its own folder. So a WordPress code lecturer can follow the student's efforts in coding. Of course, WordPress did not invent child themes for teaching

situations. The child theme is a practical solution to a serious problem. Often the theme developers manage themes. If you add changes to the original code, your edition might disappear after the next automatic update. That is why the child theme is the smart thing to do.

WordPress themes are made for tweaking. In a standard theme like Twenty Seventeen, the developer can add additional CSS on the Dashboard. A frontend developer with a solid knowledge of CSS will be able to transform the looks of a theme.

## Definition:

"A child theme is a theme that inherits the functionality and styling of another theme, called the parent theme. Child themes are the recommended way of modifying an existing theme."

Source: [Codex](#) (visited 20170301)

## How to make a Child Theme

---

A child theme is a theme that will copy the styles and layout of a theme. Of course, you could just edit the files in the directories of the original theme. But as soon as the original authors update the theme your additions could vanish. The solution is relatively simple. First, you must create a separate directory for your child theme in the `../wp-content/themes/` folder. Then you add files like **style.css**, **functions.php**, and so on.

The following pages show how a child theme is made. Two repositories have been made on Github with sample child themes in order to demonstrate the code:

- [Tw17child](#).
- [Nuit](#) - a child theme with Nodejs.

The first theme is easy to understand. The last one is an elaboration on *tw17child*. These days more and more developers work with rapid prototyping via Nodejs. The purpose of Nuit is to show the path: how to build a Childtime with *Gulp* automation, the *Bower* package manager and the *SASS* stylesheet pre-processor. These are but a few examples of the possibilities. Yet, they will show enough to get an idea about how to progress along the path of Nodejs.

But let's start with something simple:

1. Create a new folder in your WordPress directory: `/wp-content/themes/twenty-seventeen_child`
2. Create the file **style.css**

The file `style.css` will create the child theme in WordPress. The `style.css` must begin with a comment section, like this:

```
/*
Theme Name: Twenty Seventeen Child
Theme URI: https://github.com/asathoor/tw17child
Description: Twenty Seventeen Child Theme
Author: Per Thykjær Jensen
Author URI: http://multimusen.dk
Template: twentyseventeen
Version: 1.0.0
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Tags: custom-header, custom-menu-style
Text Domain: tw17child
*/

/**
 * Your styles follow here ...
 */
```

The theme should also register the stylesheet. This is accomplished in the file **functions.php** where the child theme enqueues the stylesheets from both the "parent theme" and the child theme. Here is a sample from **functions.php**:

```
function my_theme_enqueue_styles() {

    wp_enqueue_style('parent-style',
        get_template_directory_uri()
        . '/style.css');

    wp_enqueue_style('child-style',
        get_stylesheet_directory_uri()
        . '/style.css',
        array($parent_style),
        wp_get_theme()->get('Version')
    );

}

add_action('wp_enqueue_scripts', 'my_theme_enqueue_styles');
```



It is very important that you load the child-style *after* the parent theme's styles. Otherwise, the original theme may overrule your CSS rules. The function `my_theme_enqueue_style()` will add stylesheets to your head-section via `wp_enqueue_style()` functions. The `add_action( 'wp_enqueue_scripts' , 'your_function' )` will execute the function. Then WordPress will add a link to the stylesheets in the head section of your mark-up.

What you see here really is all it takes to make a child theme!

Go to your Dashboard and find the theme in Appearance. Now you can activate the child theme. If everything is ok, the website will look exactly like a Twenty Seventeen clone.

Now you can make changes to any of the files from the parent theme. Doing so is simple. If you want to change e.g. `header.php` *copy* the file from the parent theme folder to the *child theme folder*. Make whatever changes you need and save your work.

A simple proof of concept is to copy `footer.php` from the parent theme to your child theme folder. Then add something like this somewhere in **footer.php**:

```
<h1>
    <?php _e( 'Hello World' ); ?>
</h1>
```

Refresh the page in your browser. Now your creative greeting to the world should be visible in the footer of all the pages. The strange looking `_e( 'your-Text' )` will echo the "Hello World" string, and it is a preparation for internationalization of your theme. If you upload your theme to the WordPress repositories, the international translators – or Polyglotters as the translators are called in the WordPress community - will be able to translate your work.

# How to activate jQuery

---

WordPress is skipped with a version of jQuery. The Twenty Seventeen theme does not link to jQuery by default. If you want to use jQuery in your code copy [header.php](#) to your child theme folder. Add these lines before the `wp_head()` hook:

```
<!-- get jQuery (add before wp_head()) -->
<?php wp_enqueue_script("jquery"); ?>

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

WordPress will load the CSS and JavaScript files in the header. Some developers could argue that the jQuery script must be loaded towards the end of the HTML. They would have a point here. If you prefer to follow such recommendations, you can add the script via a link to a CDN or to the **jquery.js** file in **footer.php**. In this book, however, I will follow the standards of WordPress.

In order to test whether jQuery is up and running, you can try to add this script to a copy of **footer.php** in your child theme folder:

```
<script>
( function($) {

    // your code from here ...
    alert('jQuery: Alert level: green.');
```

```
    // code end

} )(jQuery); // jQuery end
</script>
```



Again – refresh the page in the browser. If you get an alert box then jQuery runs as intended. Now the possibilities are unlimited, that is within the scope of jQuery of course, now you are able to:

- Add jQueryui (but remember to enqueue an extra stylesheet).
- Work with jQuery plugins.
- Add external animation effect libraries such as Wagerfield's [Parallaxjs](#) or Artlogic's [Spritely](#).

Now we enter the creative playground of the multimedia designer...

Before we leave the subject of child themes, one last word on files and folders. Sometimes files are found in sub-folders in the parent theme. If you edit a file in a subfolder, then save the file in a similar folder structure in the child theme's folder.

One more thing: don't copy all files from the parent theme to the child theme. Just add the files you need to edit. Then your code is easier to maintain.

You can also create custom pages in a child theme. Again, you have to use the file naming conventions from the infographic [template hierarchy](#). You do it in exactly the same way as the custom pages in a theme. If it's a page, just name the page or post after the slug, e.g.: **page-SLUG.php**.

- Sample: [page-pers-galleri.php](#)

Now you are able to create a child theme. You can add the custom design to any page or post. You are able to add all the scripts, styles, and libraries you fancy.

# Nodejs and WordPress

---

By now, you should have an impression of the basic topics on themes and how to tweak them for your own purposes. The following will give you an introduction to a few advanced production tools that can speed up your work and automate certain tasks.

The first toolbox is Nodejs. Nodejs is a very popular set of tools among professional web developers.

Here is a very quick introduction to rapid WordPress prototyping with Nodejs. In the chapter you will find a case Nodejs where is used as a rapid development tool in a Twenty Seventeen child theme. But the principles also apply to themes made from scratch. Nodejs and Github are deeply integrated in numerous ways.

Let's see how you can download code from Github and then use a few Nodejs tools.

First you need to clone the theme from Github. Github is a version control repository. So far, we have seen how the development of the Twenty Seventeen theme began on Github. Before you can use Github, you'll have to install it on your system:

- Github is [available for Linux, Mac, and Windows from this uri](#).
- In order to work with Github, you have to create a user profile.
- Repositories are created on the [Github webpage](#).
- If you never used Github, [try this tutorial](#).

As soon as Github is installed, you're ready to begin. Let's get some code:

- Open a terminal window in WordPress/wp-config/themes
- Fork this repository: <https://github.com/asathoor/nuit>.
- A fork is a copy of the original theme. You can modify the code, as you like.
- Clone the code from the fork:

```
git clone https://github.com/  
YOUR-USER-NAME-HERE/nuit
```

Now Github will download the files and you have a local copy to work with. Git will download the files from the directory nuit. In the directory, you find the theme files ready for your creative work.

So now, you have the code from Github. The next step is to install Nodejs (if you have it you don't need to install it again).

But what is Nodejs?

"Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world." (quote from nodejs.org 20170307)

Install Nodejs - follow the instructions from this uri: [Download Nodejs](#).

In order to test the install, try:  
`nodejs -v`

If the terminal returns something like v4.2.6 then it's fine. Nodejs is up and running.

After Nodejs is installed, you can open the directory nuit. In the directory, you find these scripts:

- package.json
- bower.json

These files contain information about necessary software packages from the Nodejs repositories. Now it is possible to install all the Nodejs dependencies in nuit. First, run this command:

```
npm install
```

The command will download all the Nodejs dependencies for the theme. Save downloaded files in the folder node\_modules. You may open the folder, and see what's in there. One of the folders is Gulp. That's fine. We shall use Gulp later on! Npm will not download the libraries managed by Bower. But is Bower installed? Try:

```
bower -v
```

If the terminal returns something like 1.8.0 then Bower is present. If not, then you have to install Bower:

```
npm install -g bower
```

Linux, and possibly Mac, users may have to add:

```
sudo npm install -g bower
```

As soon as Bower is ready, you can install the packages:

```
bower install
```

Wait while the packages download. That's all we have to do in order to initiate the folder. Now the child theme is ready for your additions. Save the dependencies in the directories node\_modules and bower\_components.

During your theme development Bower may be used to search and install libraries, such as Bootstrap, Easeljs and similar. If you don't know the name of a relevant library, you can even try a bower search animation, which will result in a list where the word "animation" occurs. Here are a few selected samples from the search:

```
(...)  
scroll_animation https://github.com/metalabdesign/scroll_  
animation.git  
jquery-animation https://github.com/yivo/jquery-animation.git  
animations.css https://github.com/rod/animations.css.git  
animationframe https://github.com/donlion/animationFrame.git  
animationFrame https://github.com/kof/animationFrame.git  
animation.AtomicES https://github.com/AtomicES/animation.git  
(...)
```

Perhaps jquery-animation is interesting? Use Bower to install the library:

```
install jquery-animation --save
```

Remember the --save option. It will save the library to bower.json. In this way, the libraries you need will always be available. Bower integrates with Github. You can add any repository from Github:

```
bower install https://github.com/user/repo.git --save
```

You can see that Bower is a powerful tool. Especially if you're using JavaScript libraries such as Bootstrap or alike. Bower not only installs Bootstrap. Bower will also fetch dependencies. So if you install Bootstrap, then Bower will add the dependency jQuery too!

Now you have the libraries and the next step is to link to the relevant scripts and stylesheets in your theme code. How do we link them? In **functions.php** by enqueue of course.

Actually, in this case Bower has installed the css library animate.css. Your code should link to the directory. Have a [look at the function.php](#) and see how it's done:

```
wp_enqueue_style('animate-css',  
    get_stylesheet_directory_uri()  
    . '/bower_components/animate.css/animate.min.css',  
    '1.0'  
);
```

# GULP and SASS

---

Nodejs has several options for automation. A good example is the stylesheet pre-processor SASS. You write the SASS code in an ordinary editor and save the file with a .scss extension. SASS looks like CSS but there are differences. For instance you can use variables:

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

After compiling this code with `sass mySass.scss:myStyle.css`, you'd get CSS like this:

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

SASS is pretty much like working with a programming language. Variables and features such as including partials, mixins, doing math, and similar functions in the stylesheet are among the many advanced features of SASS. It is easy to translate a style tile with colours, gradients and typography to SASS variables and functions. That's the real value of SASS in rapid prototyping.

SASS can watch a directory, and compile to .css whenever you save your changes in a .scss file:

```
sass --watch input.scss:output.css.
```

More than often, you need to invoke several programs. Soon you have to start a series of programs that will compile to css, check for JavaScript errors and much more. Luckily Nodejs is able to execute such tasks automatically.

Let's have a look at GULP automation.

First check if GULP is installed with this terminal command: `gulp -v`. If the answer is "[08:10:56] CLI version 1.2.1" or similar GULP is installed. If not you should install GULP via the Nodejs package manager:

```
npm install gulp --save.
```

Since we want to work with SASS you also need the `gulp-sass` module:

```
npm install gulp-sass --save.
```

GULP will need a file called **gulpfile.js**. In this file, all your GULP tasks are defined. A sample `gulpfile.js` is found in the Nuit theme. Here are the contents of [gulpfile.js](#). The first lines of the `gulpfile` will use the `gulp` and `gulp-sass` packages:

```
'use strict';  
  
var gulp = require('gulp');  
var sass = require('gulp-sass');
```



Then two SASS related functions are created. The first function will log errors:

```
gulp.task('sass', function() {  
  return gulp.src('.sass/**/*.scss')  
    .pipe(sass().on('error', sass.logError))  
    .pipe(gulp.dest('.'));  
});
```

The second function compiles .scss to .css whenever changes are made in a file:

```
gulp.task('sass:watch', function() {  
  gulp.watch('./sass/**/*.scss', ['sass']);  
});
```

The last line of the gulpfile will execute the functions:

```
gulp.task('default', ['sass', 'sass:watch']);
```

When the gulpfile.js is ready, you can run GULP in the folder with a single command in the terminal window:

```
gulp
```

Now Gulp will listen for changes, compile when changes occur, and even alert if there are errors in your code. Have a look at the sample [gulpfile.js in the Nuit theme, here](#). Now let's try to see what happens if there are errors in your code:

1. Open a terminal window in `../themes/nuit`
2. Run the command `gulp` and see what happens
3. Edit the file and add some error in the stylesheet `../scss/style.scss`
4. Save the `.scss` file.

If you don't know how to write faulty CSS, here's an excellent error:

```
.blue {
```

As soon as `style.scss` is saved an error message is shown:

```
Error in plugin 'sass'
Message:
  sass/style.scss
Error: Invalid CSS after ".blue {":expected "}", was ""
on line 34 of sass/style.scss
>> .blue {
    -----^

[08:39:23] Finished 'sass' after 41 ms
```

The error position is pointed out. See line 34, correct the error you made on purpose. Save the file again. Now SASS is satisfied with your debugging efforts, and an error free stylesheet is compiled to style.css:

GULP will echo messages such as these in the terminal. If changes are saved in a .scss file the message 'sass:watch' will be shown. Should the script contain errors there will be an error message. SASS will only accept valid code. In this case SASS is used, but Nodejs has JavaScript linting tools and other validators. Such tests and lintings will improve the code quality. GULP is a combination of debugging and compiling.

Developing a WordPress theme or plugin with GULP will improve your code and speed up the development process, as your code will be valid.

```
[08:46:13] Starting 'sass'...  
[08:46:13] Finished 'sass' after 19 ms
```

If you feel that, the Nodejs has a steep learning curve it is not strange. Now some features are not well documented. Working in terminal windows can be daunting too. For some reason the OS designers have chosen to hide the terminal deep down in a swamp of applications. In fact, most operative systems do what they can in order to hide the tool we need.

- The Mac user can press cmd + shift + U. This will open a terminal.
- If the developer is on a Windows OS she must press the Win\_right button and type cmd. This opens up a terminal window.

**Tip:** Linux and Windows have joined at least some forces in the command line environment. From Windows 10.x you can run Bash commands in the Windows terminal. Just type bash in the terminal Window. Then you can run Linux terminal commands on Windows.

Even some Linux desktop environments have their own favourite hiding place for the terminal. In most cases, you can try alt + F2 and enter terminal in the search bar that will appear. At least this will work on Gnome, KDE, and XFCE.

Perhaps you could create a cheat sheet with terminal commands. The cheat sheet should give ideas about:

- How to navigate a theme folder.
- How to run commands such as: `npm install`, `bower install angular --save` or `gulp`.
- How to edit configuration files.
- Learning how to create a file may be a nice-to-have, e.g.:  
`touch index.php style.css functions.php` (Bash).

For those who climb the mountain to the peak of Nodejs automation then the rewards are great. You can install libraries with all dependencies with a single Bower command. Files and images will compile, minify, and save automatically.

# Innovations in WordPress 4.7.x

---

The last part of this book is a little glimpse of the future. There are many innovations in WordPress 4.7. In this chapter, I will introduce two of the most important innovations in WordPress:

- The REST API
- The wp\_cli

The REST API is a powerful innovation that will transform WordPress from a web technology to a much broader usage. From now on, you can run WordPress like a service. You can use WordPress as a backend for almost any coding language such as Python, JavaScript or Ruby.

But what is the REST API? An API is shorthand for "Application Programming Interface". In general, an API is a toolbox. The WordPress handbook gives this definition:

"The WordPress REST API provides API endpoints for WordPress data types that allow developers to interact with sites remotely by sending and receiving JSON (JavaScript Object Notation) objects." [REST API Handbook](#) visited: 20170309.

So WordPress is able to send Json strings via certain uri endpoints. What is JSON?

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language."

From: [Introducing Json](#)

JSON is a *data exchange format*. Most programming languages are able to handle JSON-objects. For example, a smartphone app written in [Lua / Corona SDK](#) would be able to fetch JSON from your WordPress site. So WordPress does not have to live in browsers any more. The API is very powerful, and the developer can write apps with features like:

- **Create** new pages or posts.
- **Read** pages, posts, categories, tags, etc.
- **Update** pages or posts.
- **Delete** content.

With the REST API the themes and even the Dashboard may be omitted. WordPress is able to exchange data with other machines:

"Content endpoints provide machine-readable external access to your WordPress site with a clear, standards-driven interface, paving the way for new and innovative methods of interacting with sites through plugins, themes, apps, and beyond."

(From the Dashboard "Welcome to WordPress 4.7.3").

# How to use the REST API

---

The WordPress REST API is a series of uri endpoints. The endpoints will return a JSON object. A JSON object is a string, formatted in a certain way. The string notation is similar to arrays or even multidimensional arrays. In programming, most languages are able to handle the JSON objects. They are integrated in JavaScript, PHP, Python, Ruby and many more will either treat the JSON string as an object or be able to transform the JSON-string to an array.

Here are a few endpoints:

- /wp/v2/posts
- /wp/v2/categories
- /wp/v2/pages

Here is a real life example, enter this in a terminal:

```
curl -i http://multimusen.dk/wp-json/wp/v2/posts
```

Now the screen should be filled with JSON. It is a very primitive proof of concept: WordPress may be used as a service for programming languages. The content of a website returns as a Json string. And most programming languages are able to integrate with Json objects. Now let's try to add REST API to a web page by JavaScript.

Add this script to a custom WordPress category page. Save the page as category-SLUG.php. A sample category page is found in the file [category-twenty-seventeen.php](#). The page follow the standards of the Twenty Seventeen theme. But the jQuery will fetch the JSON (see code on the right).

Note that the code point to: .../wp-json/wp/v2/...

The wp-json part tells WordPress that a JSON object is wanted. The JQuery will append HTML to the <div id="REST"></div>. A list with links to category pages will be added.

You can even use a similar code on a plain HTML page. What happens? You get a list of categories too! If you want posts, pages or whatever – just use the relevant URL-endpoints.

Again, we have opened yet another box of Pandora. Now you're able to develop WordPress themes without any PHP at all. You can develop your design with front-end methods, as in HTML and JavaScript.

The REST API will probably be one of the most important platforms for WordPress frontend developers in future.

```
( function($) {

    $.ajax({
        url: "<?php echo get_site_url(); ?>/wp-json/wp/v2/categories",

        // The name of the callback parameter
        jsonp: "callback",

        // Tell jQuery we're expecting JSON
        dataType: "json",

        // Tell that we want JSON
        data: {
            format: "json"
        },

        success: function(response) {
            console.log(response); // server response

            $('#REST').append('<ul>'); // append mark-up to #REST

            for(var i=0; i<response.length; i++){

                //console.log(response[i].title.rendered);
                $('#REST').append('<li> <a href="'
                    + response[i].link
                    + '">'
                    + response[i].name
                    + '</a> </li>');
            }; // li elements

        }

        $('#REST').append('</ul>'); // /ul

    }); // code end

} )(jQuery); // jquery end
```



# Introduction to the wp-cli

---

In connection with the presentation of WordPress 4.7 the founder Matt Mullenweg announced that the organization would support "... *the future of the wp-cli*" (Mullenweg, 2016). So, what on earth is the wp-cli?

## The wp\_cli is defined as:

"WP-CLI is a set of command-line tools for managing WordPress."

From: [wp-cli.org](http://wp-cli.org)

In fact, the wp\_cli is a very handy tool. The last part of this chapter is devoted to the usage of it. Normally the WordPress installation of add-ons is managed on the Dashboard. Wp-cli gives the developer the ability to create entire WordPress sites from the command line:

"wp-cli is a command-line interface that every major user of WordPress deploys and trusts. As we head into 2017, I wanted

to make sure that its future is certain for everyone who builds on it, and that the major contributors to the project, chiefly Daniel Bachhuber, are able to work on it even more in the coming year." Matt Mullenweg "[Supporting the Future of wp-cli](#)".

With Wp-cli you can:

- Install plugins
- Install themes
- Scaffold child themes
- Scaffold pages and posts
- Create content: pages, posts
- ... and more ...

Normally the average developer does not need such a tool. If you are a developer that has to mock up many WordPress sites, the wp\_cli is a powerful Swiss army knife. In fact, the wp-cli will act as if it was part of your operative system. For instance you can write scripts in Bash or Python in order to execute routine or redundant tasks.

Towards the end of 2016 Matt Mullenweg announced, that WordPress would support [Daniel Bachhuber's](#) open source project [wp-cli](#). The [wp-cli Handbook](#) is available on [make.wordpress.org](#) (visited 20170310).

So it is possible to create a script that will create pre-defined dummy content on the fly. From users to pages and posts.

The script could be written in any programming language that is able to execute operative system commands. The wp-cli may be invoked via PHP, Python, Ruby or Bash. The wp-cli script is a road to very rapid development. With a few lines of code, you can install a series of standard plugins, themes and add whatever dummy content you need.

Here is a sample. This bash script will store plugins names in an array. The *for* loop will iterate the plugin names. In these lines, each plugin is installed and activated:

```
wp plugin install $i
wp plugin activate $i
```

With a few lines of code, you can install a series of plugins. After the script, they are ready for use.

1. Create a file with this content (e.g. nano myPluginInstaller.sh):
2. Then change the file permissions to executable: `chmod a+x myPluginInstaller.sh`.
3. Execute the script `./myPluginInstaller.sh`.

### Tip:

If you work in Windows 10.x use [Bash in the terminal](#).

```
#!/bin/bash
# By: Per Thykjaer Jensen
# Purpose: install plugins
# License: GPLv3

#Plugins array
myPlugins=(
    "akismet"
    "jetpack"
    "wordpress-seo"
)

#install and activate plugins
for i in "${myPlugins[@]}"
do:
    wp plugin install $i
    wp plugin activate $i
done
```

A simple script like this demonstrates why wp-cli is such a powerful tool. Only a few lines of code are needed in order to scaffold web pages with all the standard plugins you want. With a few additions, you can make scripts that are more advanced:

- Install your multipurpose theme.
- Add whatever plugins you need.
- Create users on the fly.
- Scaffold custom pages, posts, and alike from the template hierarchy.



# Install wp-cli

The wp-cli will run on multiple operative systems. Depending on your system you can follow, these install instructions:

- [Install wp-cli on Windows](#)
- [Install wp-cli on Mac / Linux](#)

On \*NIX systems the installation is relatively simple. The following method is found in the [wp-cli Handbook](#):

1. Create a directory, and download the files there:  

```
curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar
```
2. Make the files executable.  

```
chmod a+x wp-cli.phar
```
3. Rename wp-cli.phar to wp and move it to a directory in the \$PATH:  

```
sudo mv wp-cli.phar /usr/local/bin/wp
```
4. Test the installation:  

```
wp --info
```

If everything is ok, then the terminal response looks like this:

```
PHP binary:           /user/bin/php7.0
PHP version:          7.0.15.0ubuntu0.16.10.4
php.ini used:         /etc/php/7.0/cli/php.ini
WP-CLI root dir:      phar://wp-cli.phar
WP-CLI packages dir:
WP-CLI global config:
WP-CLI project config:
WP-CLI version:       1.1.0
```

Here are a few snippets with typical wp\_cli commands:

- Get WordPress builds:  

```
wp core download --version=nightly --force
```
- List pages and posts:  

```
wp post list
```
- Create or delete pages and posts:  

```
wp post create
```
- Create or delete users:  

```
wp user delete <name>
```
- Install or delete a plugin:  

```
wp plugin install akismet
```

- Activate plugins:  

```
wp plugin activate akismet
```

The best way to learn something about wp-cli is this:  

```
wp -help <add your question here>
```

`wp --help plugins`. This command will open a help page about plugins.

In the [wp-cli Handbook](#) there are more than 100 command snippets. Here is a more complex bash sample from the handbook:

```
wp post list \
--field=ID|xargs \
-I % wp post get % \
--field=post_content|sed \
-ne 's;.*(https?g|png|gif))\).*;\1;gp'
```

According to the handbook this command will:

- List all post IDs
- Get the content
- Display image URL-s (the sed will pipe the image links)

Sometimes a plugin does not have the same install slug as it's own name. Yoast is a good example. First try to search Yoast related stuff:

wp plugin search yoast

```
Pass --path='path/to/wordpress' or run 'wp core download'.
```

```
per@Balder:~$ cd html
```

```
per@Balder:~/html$ cd wordpress/
```

```
per@Balder:~/html/wordpress$ wp plugin search yoast
```

```
Success: Showing 10 of 458 plugins.
```

name	slug	rating
Yoast SEO	wordpress-seo	96
Google Analytics for Wordpress by MonsterInsights	google-analytics-for-wordpress	78
ACF Content Analysis for Yoast SEO	acf-content-analysis-for-yoast-seo	100
Glue for Yoast SEO & AMP	glue-for-yoast-seo-amp	86
Remove Yoast SEO Comments	remove-yoast-seo-comments	100
Import Settings into WordPress SEO by Yoast	yoast-seo-settings-xml-csv-import	100
Yoast ACF Analysis	yoast-seo-acf-analysis	100
WPGlobus &#8211; Multilingual Everything!	wpglobus	96
Yoast Comment Hacks	yoast-comment-hacks	86
All Meta Stats Yoast SEO Addon	all-meta-stats-yoast-seo-addon	100

```
per@Balder:~/html/wordpress$
```

The wp-cli returns a list of Yoast related plugins. If you're lucky, one of them might be what you seek for. Perhaps you'd expect that `wp plugin install yoast` will install Yoast. In order to install the Yoast SEO with wp-cli you should type:

and press *enter*. After a moment, the plugins are installed.

The wp-cli is an excellent tool for rapid web site development. That is for those who love to get the job done fast and of course in a terminal.

```
wordpress-seo
```

# Final Words

---

This book began with an introduction to the history and management of WordPress as an organization. Part two "Diving Into Themes" is an introduction to the art of developing a WordPress front-end. Even if you find a cool and ready-made open source theme for WordPress, you should be able to tweak it via a child theme.

In the end, the purpose of this book is not to tell everything about WordPress frontend. If the odd link should guide you to even better code and tutorials, much has been accomplished.







# References

---

Asadi, Aaron, and et al. 2017. *WordPress for Beginners (New for 2017)*. The Learn IT Series.

Baym, Nancy K. 2010. *Personal Connections in the Digital Age*. Polity.

Bonnevier, Frank M. 2014. *Learning from Libraries That Use WordPress: Content Management System Best Practises and Case Studies*. The Library Quarterly.

Carney, Rob (ed.). 2014. *The Responsive Web Design Handbook*. Future Publishing Limited.

Casabona, Joe. 2014. *Responsive Design with WordPress*. New Riders.

Cloer, Jennifer. 2015. *10 Years of Git: An Interview with Git Creator Linus Torvalds*. Linux.com.

Creeber, Glen, and Royston Martin. 2009. *Digital Culture*. McGraw Hill.

Duckett, Jon. 2014. *JavaScript and JQuery*. Wiley.

Dusablon, Brian. 2014. *Responsive Performance With WordPress: Launch a Basic, yet Customizable, Content Management System with These Steps*. TD Magazine 2014:10.

Hay, Stephen. 2013. *Responsive Design Workflow*. New Riders.

Hou-Sandi, Helen. 2016. *Say Hello to Twenty Seventeen*. make.wordpress.org.

Jeffrey, Paul. 2016. *Dev Chat Summary: December 21st (4.7.1 Week 2) - 4.7 Retrospective*. make.wordpress.org.

Jensen, Per Thykjær. 2016a. *Interview med Daniel Pape (Daniel-20161115.mp3)*. The Research and Innovation department, Business Academy Aarhus.

———. 2016b. *Interview med Sandra Anastasia N. Kristholm (Sandra\_20161111.mp3)*. The Research and Innovation department, Business Academy Aarhus.

Kierkegaard, Søren. 1848. *The Point of View for My Work as an Author (1848)*.

Kristholm, Sandra Anastasia N. 2016. *Aarhus "I et øjeblik"*. Dissertation, Business Academy Aarhus, Multimedia Ddesign.

Marcotte, Ethan. 2010. *Responsive Web Design*. A List Apart.

Mead, Margaret. 1928. *Coming of Age in Samoa - A Psychological Study of Primitive Youth for Western Civilisation*. William Morrow & Company.

Mullenweg, Matt. 2016. *Supporting the Future of Wp-Cli*. [make.wordpress.org](http://make.wordpress.org).

Raymond, Eric S. 1997. *The Cathedral and the Bazaar*.

Schein, Edgar H. 2010. *Organizational Culture and Leadership*. Jossey-Bass - A Wiley Imprint.

Tolkien, J.R.R. 1954. *The Lord of the Rings*. George Allen & Unwin (UK).

wordpress.org. 2010. *Twenty Ten*. [wordpress.org](http://wordpress.org).

———. 2011. *Twenty Eleven*. [wordpress.org](http://wordpress.org).

———. 2012. *Twenty Twelve*. [wordpress.org](http://wordpress.org).

———. 2013. *Twenty Thirteen*. [wordpress.org](http://wordpress.org).

———. 2014. *Twenty Fourteen*. [wordpress.org](http://wordpress.org).

———. 2015. *Twenty Fifteen*. [wordpress.org](http://wordpress.org).

———. 2016. *Twenty Sixteen*. [wordpress.org](http://wordpress.org).

———. 2017. *Twenty Seventeen*. [wordpress.org](http://wordpress.org).

