WordPress & Hosting Techniques

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: wordpress

It is an unofficial and free WordPress ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official WordPress.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with WordPress

Remarks

WordPress is an open source Content Management System (CMS) which is used to build and manage websites. WordPress is the most popular CMS on the internet by a country mile, powering about half of all CMS websites at time of writing and about a quarter of all websites on the internet.

WordPress started life as a platform for blogging but has evolved over the years to be suitable for most types of websites. The interface can be used without coding knowledge making it popular for beginners and developers who want to empower their clients to manage their own website.

Another large factor in the popularity of WordPress is it's flexibility, mostly due to the core's plugin and theming systems. The plugin system makes it easy to extend the core functionality without modifying the core code. In a similar manner, the theming system makes it easy to change the website's layout and aesthetics. There are now thousands of free and premium WordPress plugins and themes available. Many of these are located at the wordpress.org plugin repository and theme repository respectively.

WordPress is developed by it's own community, but is strongly associated with the company Automattic, which employs many of WordPress' core developers.

Code

WordPress is built upon the PHP server scripting language and the MySQL querying language. WordPress uses MySQL as a datastore for user content and configuration. The PHP wrangles the content data into a HTML webpage with all the necessary assets.

wordpress.com vs wordpress.org

You can use WordPress by signing up for Automattic's wordpress.com service and hosting your website on their servers, or you can download the WordPress software from wordpress.org and host your website on a server under your control. The first option is easy but you cannot edit any site code. You can only make changes through the WordPress interface. The second option requires more work but gives you flexibility to do whatever you like with your website code. If you are a StackOverflow user, you probably will be going with the second option.

Open Source

WordPress is open source software meaning it is free to use and anyone can view the source code and contribute to it. Potential contributors can get started by reading the Contribution page of the WordPress codex..

Bugs can be reported by submitting a bug on the WordPress ticket tracker.

Documentation

WordPress is officially documented in the WordPress Codex at WordPress.org. Developers working with WordPress will be particularly interested in the Developer Codex section and Developer Reference section of wordpress.org.

Versions

Version	Release Date
1.0	2004-01-03
1.2	2004-05-22
1.5	2005-02-17
2.0	2005-12-26
2.1	2007-01-22
2.2	2007-05-16
2.3	2007-09-24
2.5	2008-03-29
2.6	2008-07-15
2.7	2008-12-10

Version	Release Date
2.8	2009-06-10
2.9	2009-12-18
3.0	2010-06-17
3.1	2011-02-23
3.2	2011-07-04
3.3	2011-12-12
3.4	2012-06-13
3.5	2012-12-11
3.6	2013-08-01
3.7	2013-10-24
3.8	2013-12-12
3.9	2014-04-16
4.0	2014-09-04
4.1	2014-12-17
4.2	2015-04-23
4.3	2015-08-18
4.4	2015-12-08
4.5	2016-04-12
4.6	2016-08-16
4.7	2016-12-06
4.8	2017-06-08

Examples

Introduction to WordPress

WordPress [WP] is an open source Content Management System for building apps, websites, and blogs. WP is written in PHP and uses MySQL as the data store for the user content and

configuration. It has a rich ecosystem of plugins and themes and enjoys a vibrant open source community, good documentation, and low barriers to entry. Usability and developer documentation can be found in the WP Codex.

A part of WordPress that makes it different from most other CMS products is its Event Driven Programming. This is a different way of programming and logic representation then the MVC (Model View Controller) architecture which is used by most of the CMS systems. WordPress uses the concepts of Actions and Filters. They form a queue of events that allow plugins and themes to insert, modify or even remove parts of the final web application page and/or parts. A similar concept is JIT or Just-In-Time compiling.

While historically WordPress has been known as a blogging platform, and it may never lose this stigma, the focus of the core WordPress team has clearly changed. With the 2016 State of the Word, by founder Matthew Mullenweg, we can see a clear shift in goals, vision and effort. In 2016, we saw amazing progress when the WordPress core adopted a majority of the very popular REST API plugin. This was clearly an intention of the core team early on when they began a bold effort of building a front-end JavaScript CMS admin panel, that breaks away from the golden standard we have seen for so many years; they called it Calpyso.

WordPress Themes

Mapping URLs to specific templates

To fully grasp WordPress themes, you must understand two primary concepts:

1. Permalinks
2. The Template Hierarchy

A permalink is a permanent, non-changing URL (or link, to a specific resource. For instance:

- example.com/about-us/ (a Page in WP)
- example.com/services/ (a listing of multiple items, also called an "archive" in WP lingo)
- example.com/services/we-can-do-that-for-you/ (an individual item)

When a user requests a URL, WordPress reverse-engineers the permalink to figure out which template should control its layout. WordPress looks for the various template files that *could* control this particular piece of content, and ultimately gives preference to the most specific one it finds. This is known as the Template Hierarchy.

Once WP finds the matching view template in the hierarchy, it uses that file to process and render the page.

For example: `index.php` (the default, "catch-all" template) will be overridden by `archive.php` (the default template for list-based content), which will in turn be overridden by `archive-services.php` (a template file specifically for the archive named "services").

Here is a great visual reference for the Template Hierarchy

Basic Theme Directory Structure

A simple theme looks something like this:

```
    // Theme CSS
    style.css

    // Custom functionality for your theme
    functions.php

    // Partials to include in subsequent theme files
    header.php
    footer.php
    sidebar.php
    comments.php

    // "Archives", (listing views that contain multiple posts)
    archive.php
    author.php
    date.php
    taxonomy.php
    tag.php
    category.php

    // Individual content pages
    // Note that home and frontpage templates are not recommended
    // and they should be replaced by page templates
    singular.php
    single.php
    page.php
    front-page.php
    home.php

    // Misc. Utility Pages
    index.php (a catch-all if nothing else matches)
    search.php
    attachment.php
    image.php
    404.php
```

Example of a "Single" (template for an individual post)

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

<?php if ( have_posts() ) while ( have_posts() ) : the_post(); ?>
    <h1><?php the_title(); ?></h1>
    <?php the_content(); ?>
    <?php comments_template( '', true ); ?>
<?php endwhile; ?>

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

What's happening here? First, it loads `header.php` (similar to a PHP include or require), sets up The Loop, displays `the_title` and `the_content`, then includes `comments.php`, `sidebar.php`, and `footer.php`. The Loop does the heavy lifting, setting up a `Post` object, which contains all the information for the currently-viewed content.

Example of an "Archive" (template for a list of multiple posts)

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

<?php if ( have_posts() ) while ( have_posts() ) : the_post(); ?>
    <a href="<?php the_permalink(); ?>"<?php the_title(); ?></a>
    <?php the_excerpt(); ?>
<?php endwhile; ?>

<?php
    next_posts_link( 'Older Entries', $the_query->max_num_pages );
    previous_posts_link( 'Newer Entries' );
?>

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

First, it includes `header.php`, sets up The Loop, and includes `sidebar.php`, and `footer.php`. But in this case there are multiple posts in the loop, so instead an excerpt is shown with a link to the individual post. `next_posts_link` and `previous_posts_link` are also included so the archive can paginate results.

Posts, Pages, Custom Post Types, and Custom Fields

Out of the box, WordPress supports two types of content: `Posts` and `Pages`. Posts are typically used for non-hierarchical content like blog posts. Pages are used for static, standalone content like an About Us page, or a company's Services page with nested sub-pages underneath.

As of version 3.0, developers can define their own custom post types to extend the functionality of WordPress beyond just the basics. In addition to custom post types, you can also create your own custom fields to attach to your posts/pages/custom post types, allowing you to provide a structured way of adding and accessing metadata within your templates. See: Advanced Custom Fields.

Chapter 2: Actions and Filters

Syntax

- add_action(tag, function_to_call, priority, num_of_args);
- add_filter(tag, function_to_call, priority, num_of_args);

Parameters

Parameter	Explanation
$tag	(string) (Required) The name of the action to which the $function is hooked.
$function	(callable) (Required) Requires a string containing the function name or anonymous function. See examples for adding functions within classes.
$priority	(int) default = 10. Functions attached to hooks/ filters will run in the priority assigned. You may have a situation where you want to work with code before any other actions, set priority =1 or after all other attached functions priority = 100 etc. As with all php functions, you can use the function without passing a value for a variable where a default value has been set, but if you wish to change the number of parameters returned, you must specify!
$parameters	(int) default = 1. The number of parameters returned to your attached function. The parameters returned will depend on the number attached where the hook was created. See `apply_filters()` and `do_action()` for more details.

Remarks

Wordpress Hooks

Something that often confuses developers when starting to work with WordPress are the use of `apply_filters()` and `add_action()`. You will often see plugins/themes making use of these in code and if you don't understand the concept, you will find it hard to work with them.

In brief (very brief, look up WordPress load flowchart for process in detail), WordPress loads in the following way:

1. wp-load.php - functions etc
2. mu-plugins - any files found in the mu-plugins folder - often used to serve cached objects
3. Plugins - no particular order, any installed and activated plugins will be loaded
4. Active child theme / parent theme
5. init - rest of data
6. template

If you are a developer and working with a functions file, you can see both are loaded earlier in the process than the files you are working with. Meaning you can't modify processes (note you cannot overwrite functions) or variables that run later or have not been defined yet. Also theme developers may place hooks in their code to allow plugins to hook to or plugins might allow for other plugins to overwrite their variables. Now this may be confusing thus far, but hang in there.

To understand `add_filter()` and `add_action()` we need to look at how the hooks are created in the first place.

```
$arga= 'hello';
do_action('im_a_hook', $arga );
```

When you encounter the above in WordPress, it will call any functions attached to the hook `im_a_hook` (look up `$wp_filter` for information on the process). In your attached function `$arga` will be available for the attached function to work with.

```
add_action('im_a_hook', 'attached_function');

function attached_function($arga){
     echo $arga;
}
```

This opens up powerful new opportunities to modify variables at certain points of the load process. Remember we said earlier that templates are loaded after plugins/ themes? One common plugin is WooCommerce which creates screens later in the process, I'm not going to document how but an example of `do_action` can be found in the plugin.

```
do_action( 'woocommerce_after_add_to_cart_button' );
```

Here we have a hook created that passes no variables back, but we can still have fun with it:

```
add_action( 'woocommerce_after_add_to_cart_button', 'special_offer');

function special_offer(){
    echo '<h1>Special Offer!</h1>;
}
```

The above `add_action` will `echo` a heading of special offer where `do_action('woocommerce_after_add_to_cart_button')` is located which is when creating a WooCommerce screen. So we can use this hook to insert html. Other uses might include redirecting to a different screen altogether, etc.

Also multiple variables may be passed to the function. Try this in your themes functions. Note the last parameter we are setting to 3, because we want to work with the 3 available parameters. If we changed this to 2, only 2 would be returned and we would get a undefined error.

```
add_action('custom_hook', 'attached_function', 10, 3);

function attached_function($a,$b,$c){
```

```
    var_dump($a);
    var_dump($b);
    var_dump($c);

}


$arga = 1;
$argb = 2;
$argc = 3;

do_action('custom_hook', $arga, $argb, $argc);
exit;
```

There is another WP hook type called a filter. A filter is different from an action in its usage, an action can only receive variables, obviously these variables are within the functions scope (you should know what php scope is, if not google). Filters pass back the returned data, so you can use to modify variables.

```
$filter_me= apply_filters('im_a_filter', $variable_to_filter);
```

Where you see the above, you can modify the value of `$filter_me` as any data you return will be the value stored in the variable. So for example (note we are changing `$variable_to_filter` to `$filter_me` in the example):

```
add_filter('im_a_filter', 'attached_function', 100);

function attached_function($filter_me){

    $filter_me= 'ray';

    return $filter_me;

}


$filter_me = 'bob';
$filter_me= apply_filters('im_a_filter', $filter_me);
```

The `$filter_me` variable will now contain *'ray'* rather than *'bob'*, we have set a priority of 100 so we are reasonably confident no one is changing the value after use (there can be multiple filters running on the same hook) So we can now change variables used later in the process if `apply_filters()` is present.

You can also pass multiple parameters, but you can only change the value of one. You must also return a value, or else your variable will contain nothing. If you understand how you use php to assign values/arrays/objects to variables this will be obvious to you, e.g.:

```
add_filter('im_a_filter', 'attached_function', 100, 3);

function attached_function($filter_me, $arga, $argb){

    $filter_me= 'ray'.$arga.$argb;
```

```
    $arga= 'you fool';

    return $filter_me;

}

$filter_me = 'bob';

$arga = ' middlename';
$argb = ' surname';

$filter_me= apply_filters('im_a_filter', $filter_me, $arga, $argb);
```

The `$filter_me` variable now contains *'ray middlename surname'*. But what about `$arga`? This still contains *'middlename'*, changing an `$arga` to *'you fool'* within our function has no effect on the defined value outside of its scope (there are ways, google globals etc.)

add_action($hook_name, $function, $priority, $parameters)

add_filter($hook_name, $function, $priority, $parameters);

Examples

add_action - init

```
add_action('init', 'process_post');

function process_post(){
   if($_POST)
     var_dump($_POST);
}
```

add_action - init - anonymous function

```
add_action('init' , function(){
    echo 'i did something';
});
```

add_action - init - within class object

```
 class sample{

     public function __construct(){
         add_action('init', array($this, 'samp') );
     }


     public function samp(){ // must be public!!
         echo 'i did something';
     }
 }

 new sample();
```

add_action - init - within static class

```
class sample{

    public static function add_action_func(){
        //note __CLASS__ will also include any namespacing
        add_action('init', array(__CLASS__, 'samp') );
    }

    public static function samp(){
        echo 'i did something';
    }

}

sample::add_action_func();
```

Chapter 3: Add Shortcode

Syntax

- `add_shortcode( $tag , $func );`

Parameters

Parameter	Details
$tag	*(string) (required)* Shortcode tag to be searched in post content
$func	*(callable) (required)* Hook to run when shortcode is found

Remarks

- The shortcode callback will be passed three arguments: the shortcode attributes, the shortcode content (if any), and the name of the shortcode.
- There can only be one hook for each shortcode. Which means that if another plugin has a similar shortcode, it will override yours or yours will override theirs depending on which order the plugins are included and/or ran.
- Shortcode attribute names are always converted to lowercase before they are passed into the handler function. Values are untouched.
- Note that the function called by the shortcode should never produce output of any kind. Shortcode functions should return the text that is to be used to replace the shortcode. Producing the output directly will lead to unexpected results. This is similar to the way filter functions should behave, in that they should not produce expected side effects from the call, since you cannot control when and where they are called from.

Examples

Simple shortcode for recent post

`add_shortcode` is wp keyword.

```
// recent-posts is going to be our shortcode.
add_shortcode('recent-posts', 'recent_posts_function');

// This function is taking action when recent post shortcode is called.
function recent_posts_function() {
   query_posts(array('orderby' => 'date', 'order' => 'DESC' , 'showposts' => 1));
   if (have_posts()) :
      while (have_posts()) : the_post();
         $return_string = '<a href="'.get_permalink().'">'.get_the_title().'</a>';
      endwhile;
```

```
   endif;
   wp_reset_query();
   return $return_string;
}
```

This snippet can be placed in your theme `functions.php`.

`[recent-posts]` This is out shortcode for recent post. We can apply this shortcode in backend (such as pages, post, widgets).

We can also used same shortcode inside our code. with the help of `do_shortcode`.
Eg. **`echo do_shortcode( '[recent-posts]' );`**

Advanced shortcode for recent posts

This functions takes parameter for how many recent posts you want to display.

Ex : you want to display only five recent posts. Just passed the arguments with posts="5" (you can pass any number of recent posts that you want to display).

Function fetch only five recent posts from database.

```
// recent-posts is going to be our shortcode.
add_shortcode('recent-posts', 'recent_posts_function');

// Functions takes parameter such as posts="5".
function recent_posts_function($atts){
   extract(shortcode_atts(array(
      'posts' => 1,
   ), $atts));

   $return_string = '<ul>';
   query_posts(array('orderby' => 'date', 'order' => 'DESC' , 'showposts' => $posts));
   if (have_posts()) :
      while (have_posts()) : the_post();
         $return_string .= '<li><a href="'.get_permalink().'">'.get_the_title().'</a></li>';
      endwhile;
   endif;
   $return_string .= '</ul>';

   wp_reset_query();
   return $return_string;
}
```

Eg. **`echo do_shortcode( '[recent-posts posts="5"]' );`**

Chapter 4: Add/remove contact info for users with user_contactmethods filter hook

Examples

Enabling most popular social networks

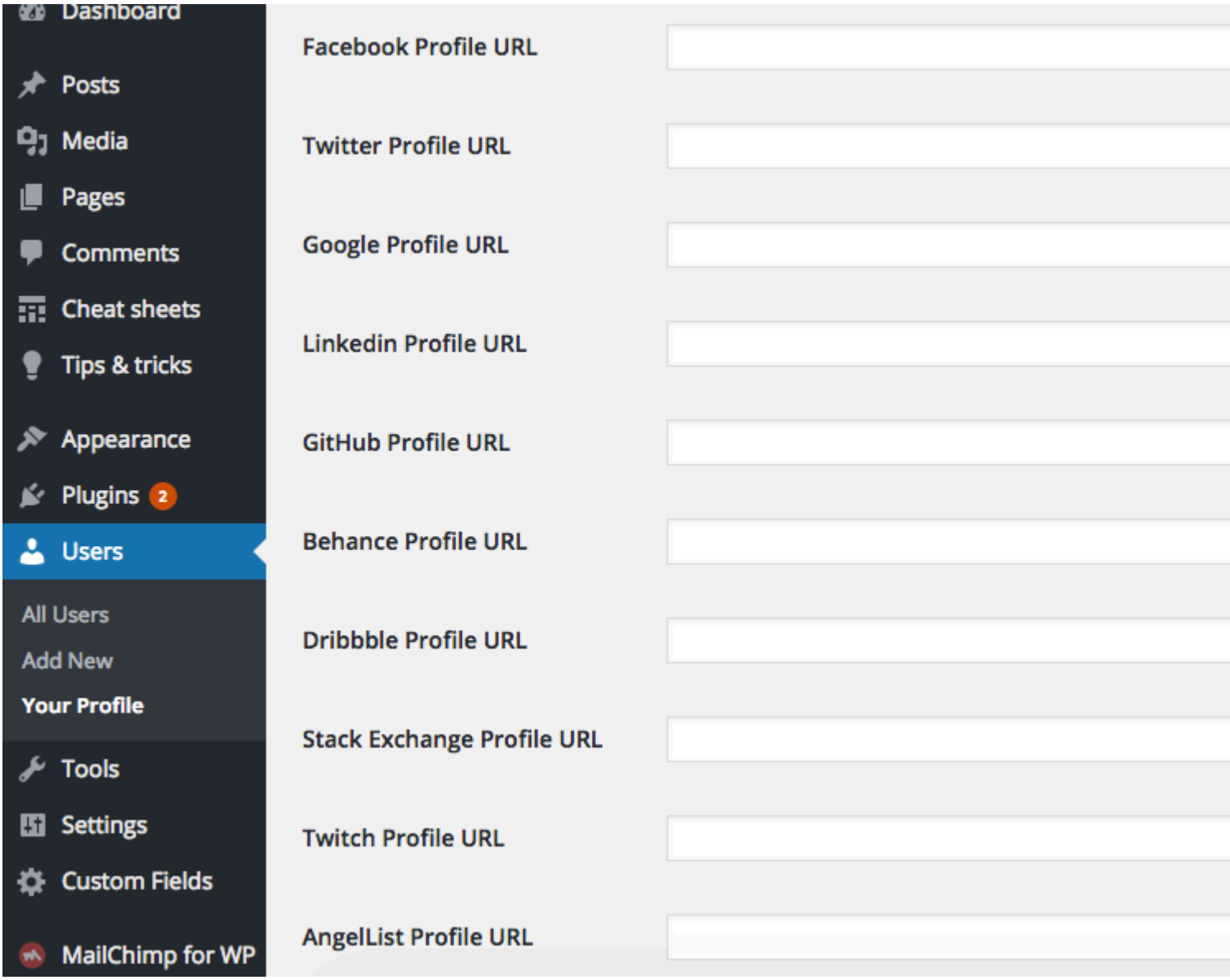
```
function social_profiles( $contactmethods ) {

    $contactmethods['facebook_profile']  = 'Facebook Profile URL';
    $contactmethods['twitter_profile']   = 'Twitter Profile URL';
    $contactmethods['google_profile']    = 'Google Profile URL';
    $contactmethods['linkedin_profile']  = 'Linkedin Profile URL';
    $contactmethods['github_profile']    = 'GitHub Profile URL';
    $contactmethods['behance_profile']   = 'Behance Profile URL';
    $contactmethods['dribbble_profile']  = 'Dribbble Profile URL';
    $contactmethods['stack_profile']     = 'Stack Exchange Profile URL';
    $contactmethods['twitch_profile']    = 'Twitch Profile URL';
    $contactmethods['angellist_profile'] = 'AngelList Profile URL';

    return $contactmethods;
}

add_filter( 'user_contactmethods', 'social_profiles', 10, 1);
```

You will get this fileds in your dashboard:



And this is how you retrieve it in code

```
<?php $user_stack_exchange = get_the_author_meta( 'stack_profile' ); ?>
```

Removing contact method

```
function remove_contact_methods( $contactmethods ) {

    unset($contactmethods['facebook_profile']);
    unset($contactmethods['twitter_profile']);

    return $contactmethods;
}

add_filter( 'user_contactmethods', 'remove_contact_methods', 10, 1);
```

Chapter 5: add_action()

Syntax

- add_action($tag, $function_to_add)
- add_action($tag, $function_to_add, $priority)
- add_action($tag, $function_to_add, $priority, $accepted_args)

Parameters

Parameter	Details
$tag	*(string)* The name of the action to which the procedure `$function_to_add` will be hooked.
$function_to_add	*(callable)* The callable function/procedure you want to be called.
$priority	*(int)* The priority level to which the `$function_to_add` will be executed. Optional. Default 10.
$accepted_args	*(int)* The number of arguments that the callable function `$function_to_add` accepts. Optional. Default 1.

Examples

Direct function callback

```
add_action( 'init', function() {
    // do something here
} );
```

Using a function block to hook a set of instructions. With the `init` hook, the set of instructions will be executed right after wordpress has finished loading the necessary components.

Function name reference callback

```
function my_init_function() {
    // do something here
}

add_action( 'init', 'my_init_function' );
```

Using the name of the function to hook a set of instructions. With the `init` hook, the set of instructions will be executed right after wordpress has finished loading the necessary components.

Class static method callback

```
class MyClass {
    static function my_init_method() {
        // do something here
    }
}

add_action( 'init', array( 'MyClass', 'my_init_method' ) );
```

Using a static method of a class to hook a set of instructions. With the `init` hook, the set of instructions will be executed right after wordpress has finished loading the necessary components.

Object method callback

```
class MyClass {
    function my_init_method() {
        // do something here
    }
}

$obj = new MyClass();

add_action( 'init', array( $obj, 'my_init_method' ) );
```

Using a method of an object to hook a set of instructions. With the `init` hook, the set of instructions will be executed right after wordpress has finished loading the necessary components.

Chapter 6: add_editor_style()

Introduction

The function allows user to load stylesheets for the TinyMCE editor

Syntax

- add_editor_style($stylesheet)

Parameters

Parameter	Details
$ stylesheet	(array or string) (Optional) Stylesheet name or array thereof, relative to theme root. Defaults to 'editor-style.css'

Examples

Loading a single css file

Code

```
function add_new_style() {
    add_editor_style( 'file-name-here.css' );
}
add_action( 'admin_init', 'add_new_style' );
```

Explanation

In the above code, we used add_editor_style to load the css file. We also used add_action to make sure that WordPress runs our function.

Chapter 7: add_menu_page()

Introduction

This function is to add an item in the admin panel's nav bar.

Syntax

- add_menu_page($page_title, $menu_title, $capability, $menu_slug, $function, $icon_url, $position)

Parameters

Parameter	Details
$page_title	(string) The text to be displayed in the title tags of the page when the menu is selected.
$menu_title	(string) The text to be used for the menu.
$capability	(string) The capability required for this menu to be displayed to the user.
$menu_slug	(string) The slug name to refer to this menu by (should be unique for this menu).
$function	(callable) (optional) The function to be called to output the content for this page.
$icon_url	(string) (optional) The URL to the icon to be used for this menu.
$position	(int) (optional) The position in the menu order this one should appear.

Remarks

Here is a list of the default positions (for $position)

- 2 – Dashboard
- 4 – Separator
- 5 – Posts
- 10 – Media
- 15 – Links
- 20 – Pages
- 25 – Comments
- 59 – Separator

- 60 – Appearance
- 65 – Plugins
- 70 – Users
- 75 – Tools
- 80 – Settings
- 99 – Separator

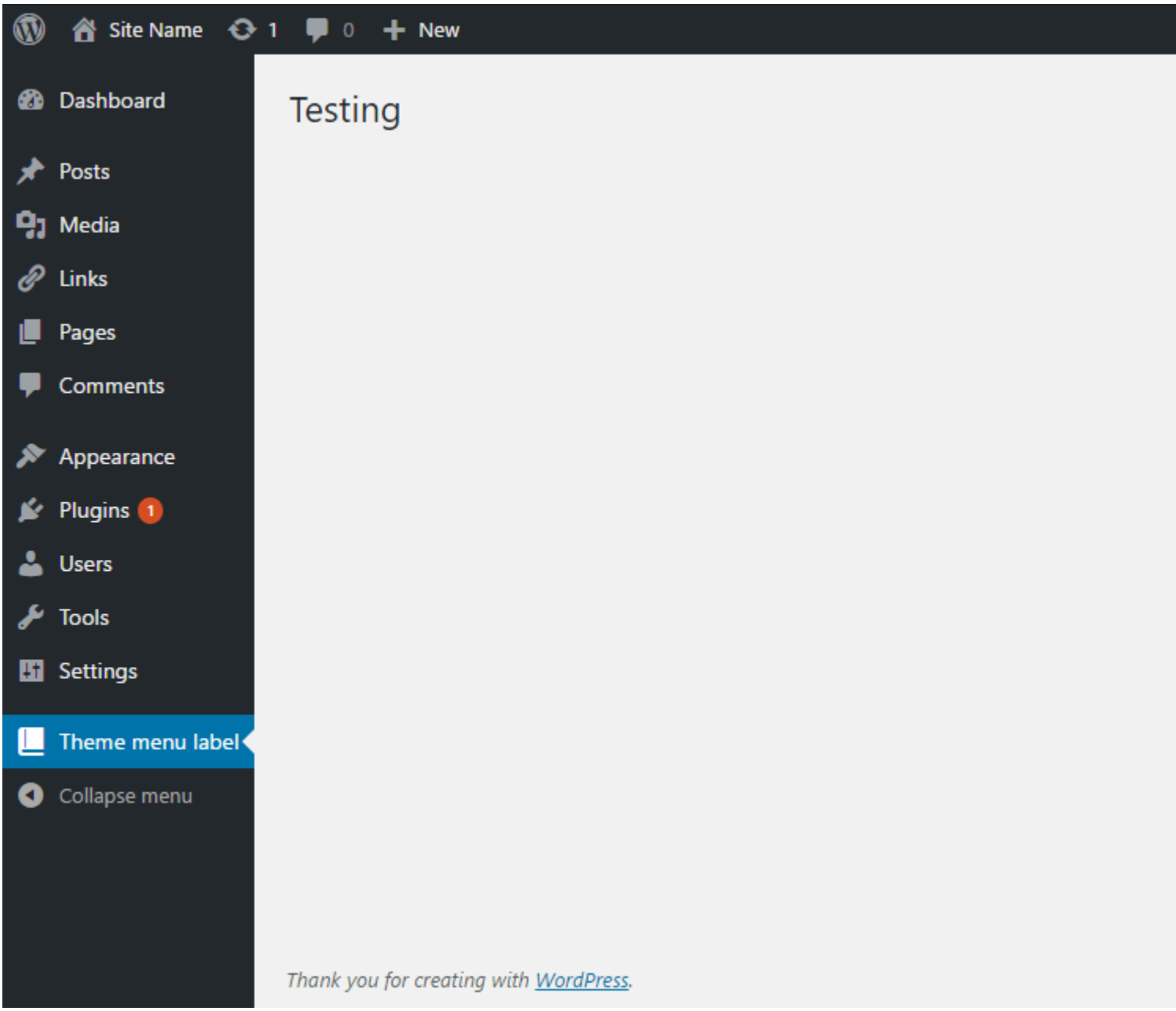
Examples

Adding the "Theme page title" item to the nav bar

Code

```
function add_the_theme_page(){
    add_menu_page('Theme page title', 'Theme menu label', 'manage_options', 'theme-options',
'page_content', 'dashicons-book-alt');
}
add_action('admin_menu', 'add_the_theme_page');
function page_content(){
    echo '<div class="wrap"><h2>Testing</h2></div>';
}
```

Output



Explanation

In the code, we created a function named `add_the_theme_page` and we used `add_menu_page` to add the item to the navbar. Please check the parameters part in this page to know about the arguments we passed in. Then we used `add_action` to run our `add_the_theme_page` function. Finally, we created the function `page_content` to display contents in the page.

OOP & how to load scripts/styles on menu page

```
<?php
/*
 *  Plugin Name: Custom Admin Menu
 */

class SO_WP_Menu {

    private $plugin_url;
```

```
    public function __construct() {
        $this->plugin_url = plugins_url( '/', __FILE__ );
        add_action( 'plugins_loaded', array( $this, 'init' ) );
    }

    public function init() {
        add_action( 'admin_menu', array( $this, 'add_menu' ) );
    }

    public function add_menu() {
        $hook = add_menu_page(
            'My Menu',                  // Title, html meta tag
            '<span style="color:#e57300;">My Menu</span>', // Menu title, hardcoded style
            'edit_pages',               // capability
            'dummy-page-slug',          // URL
            array( $this, 'content' ), // output
            null,                       // icon, uses default
            1                           // position, showing on top of all others
        );
        add_action( "admin_print_scripts-$hook", array( $this, 'scripts' ) );
        add_action( "admin_print_styles-$hook", array( $this, 'styles' ) );
    }

    public function content() {
        ?>
            <div id="icon-post" class="icon32"></div>
            <h2>Dummy Page</h2>
            <p> Lorem ipsum</p>
        <?php
    }

    # Printing directly, could be wp_enqueue_script
    public function scripts() {
        ?><script>alert('My page');</script><?php
    }

    # Enqueing from a CSS file on plugin directory
    public function styles() {
        wp_enqueue_style( 'my-menu', $this->plugin_url . 'my-menu.css' );
    }
}

new SO_WP_Menu();
```

What's important to note in this example is that, when using `add_menu_page()`, it returns a hook that can be used to target our exact page and load Styles and Scripts there.
A common mistake is to enqueue without targeting and that spills scripts and styles all over `/wp-admin`.
Using OOP we can store common variables to be used among internal methods.

Chapter 8: add_submenu_page()

Introduction

This function is to add a sub-item to an existing item in the admin panels nav bar.

Syntax

- add_submenu_page($parent_slug, $page_title, $menu_title, $capability, $menu_slug, $function)

Parameters

Parameter	Details
$parent_slug	(string) The slug name for the parent menu (or the file name of a standard WordPress admin page).
$page_title	(string) The text to be displayed in the title tags of the page when the menu is selected.
$menu_title	(string) The text to be used for the menu.
$capability	(string) The capability required for this menu to be displayed to the user.
$menu_slug	(string) The slug name to refer to this menu by (should be unique for this menu).
$function	(callable) (Optional) The function to be called to output the content for this page.

Remarks

Here are a list of slugs for $parent_slug

- Dashboard: ‘index.php’
- Posts: ‘edit.php’
- Media: ‘upload.php’
- Pages: ‘edit.php?post_type=page’
- Comments: ‘edit-comments.php’
- Custom Post Types: ‘edit.php?post_type=your_post_type’
- Appearance: ‘themes.php’
- Plugins: ‘plugins.php’
- Users: ‘users.php’

- Tools: ‘tools.php’
- Settings: ‘options-general.php’
- Network Settings: ‘settings.php’

Examples

Adding the "Submenu Page" as a sub-page of "Tools" to the nav bar

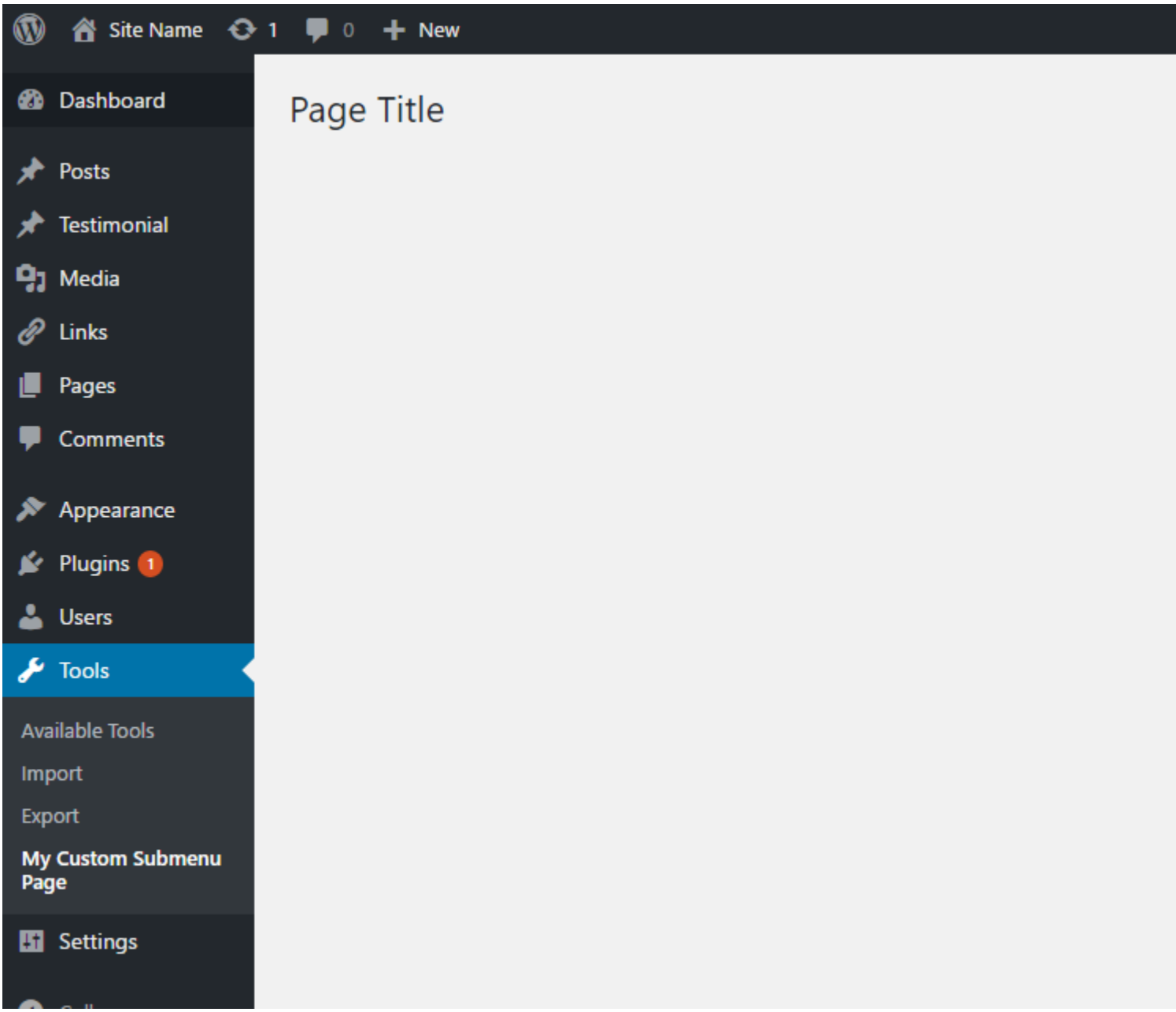
Code

```
add_action('admin_menu', 'register_my_custom_submenu_page');

function register_my_custom_submenu_page() {
    add_submenu_page(
        'tools.php',
        'Submenu Page',
        'My Custom Submenu Page',
        'manage_options',
        'my-custom-submenu-page',
        'my_custom_submenu_page_content' );
}

function my_custom_submenu_page_content() {
    echo '<div class="wrap">';
        echo '<h2>Page Title</h2>';
    echo '</div>';
}
```

Output



Explanation

In the code, we created a function named `register_my_custom_submenu_page` and we used `add_submenu_page` to add the item to the navbar as a child of tools.php, which is the Tools page.

Please check the parameters part in this page to know about the arguments we passed in. Then we used `add_action` to run our `register_my_custom_submenu_page` function. Finally, we created the function `my_custom_submenu_page_content` to display contents in the page.

Chapter 9: add_theme_support()

Introduction

This function registers features that the theme supports.

Syntax

- add_theme_support($feature)

Parameters

Parameter	Details
$feature	(string) The feature being added.

Remarks

List of features to be used in $feature:

- 'post-formats'
- 'post-thumbnails'
- 'html5'
- 'custom-logo'
- 'custom-header-uploads'
- 'custom-header'
- 'custom-background'
- 'title-tag'
- 'starter-content'

Examples

Adding theme support for post formats

```
add_theme_support( 'post-formats', array( 'formatone', 'formattwo' ) );
```

Adding theme support for post thumbnails to posts

```
add_theme_support( 'post-thumbnails', array( 'post' ) );
```

The above code only allows post thumnails on all posts. To enable the feature on all post types, do:

```
add_theme_support( 'post-thumbnails' );
```

Chapter 10: Admin Dashboard Widgets

Introduction

With an admin dashboard widget you are able to display any kind of information at the admin dashboard. You can make multiple widgets if you want. You can add the code to the functions.php of your theme or to your plugin.

Syntax

- add_action($tag, $function_to_add, $priority, $accepted_args);
- wp_add_dashboard_widget($widget_id, $widget_name, $callback, $control_callback, $callback_args);

Parameters

Parameter	Details
$tag	(*string required*) Name of the action where $function_to_add is hooked
$function_to_add	(*callable required*) Name of the function you want to call.
$priority	(*int optional*) Place of the function call in all functions of action (default = 10)
$accepted_args	(*int optional*) Number of parameters the function accepts (default = 1)
$widget_id	(*string required*) Unique slug for your widget
$widget_name	(*string required*) Name of your widget (displayed in the head)
$callback	(*callable required*) Name of the function which displays die content of your widget
$control_callback	(*callable optional*) Name of the function which handles the widget options forms
$callback_args	(*array optional*) Parameters of the $control_callback function

Examples

Simple widget (displays text)

This will add a simple widget which displays just a small message.

```
 add_action('wp_dashboard_setup', 'register_my_dashboard_widgets');

 function register_my_dashboard_widgets() {
     wp_add_dashboard_widget('myInfo_widget', 'Important Information', 'display_infoWidget');
 }

function display_infoWidget() {
     echo '<p>At the first of february this site gets a new design.
     Therefore is wont be available this day. To see the current progress you can visit
     <a href="http://www.justanexample.com" >this site</a></p>';
 }
```

Chapter 11: AJAX

Examples

AJAX request with a JSON response

functions.php:

```
// We add the action twice, once for logged in users and once for non logged in users.
add_action( 'wp_ajax_my_action', 'my_action_callback' );
add_action( 'wp_ajax_nopriv_my_action', 'my_action_callback' );

// Enqueue the script on the front end.
add_action( 'wp_enqueue_scripts', 'enqueue_my_action_script' );
// Enqueue the script on the back end (wp-admin)
add_action( 'admin_enqueue_scripts', 'enqueue_my_action_script' );

function my_action_callback() {
    $json = array();

    if ( isset( $_REQUEST['field2'] ) ) {
        $json['message'] = 'Success!';
        wp_send_json_success( $json );
    } else {
        $json['message'] = 'Field 2 was not set!';
        wp_send_json_error( $json );
    }
}

function enqueue_my_action_script() {
    wp_enqueue_script( 'my-action-script', 'path/to/my-action-script.js', array( 'jquery' ),
null, true );
    wp_localize_script( 'my-action-script', 'my_action_data', array(
        'ajaxurl' => admin_url( 'admin-ajax.php' ),
    ) );
}
```

my-action-script.js:

```
(function( $ ) {
    'use strict';

    $( document ).on( 'ready', function() {
        var data = {
            action: 'my_action',
            field2: 'Hello World',
            field3: 3
        };

        $.getJSON( my_action_data.ajaxurl, data, function( json ) {
            if ( json.success ) {
                alert( 'yes!' );
            } else {
                alert( json.data.message );
            }
```

```
        } );
    } );

})( jQuery );
```

AJAX with .ajax() and WordPress Nonce

functions.php

```
//Localize the AJAX URL and Nonce
add_action('wp_enqueue_scripts', 'example_localize_ajax');
function example_localize_ajax(){
    wp_localize_script('jquery', 'ajax', array(
        'url' => admin_url('admin-ajax.php'),
        'nonce' => wp_create_nonce('example_ajax_nonce'),
    ));
}

//Example AJAX Function
add_action('wp_ajax_example_function', 'example_function');
add_action('wp_ajax_nopriv_example_function', 'example_function');
function example_function(){
    if ( !wp_verify_nonce($_POST['nonce'], 'example_ajax_nonce') ){
        die('Permission Denied.');
    }

    $firstname = sanitize_text_field($_POST['data']['firstname']);
    $lastname = sanitize_text_field($_POST['data']['lastname']);

    //Do something with data here
    echo $firstname . ' ' . $lastname; //Echo for response
    wp_die(); // this is required to terminate immediately and return a proper response:-
https://codex.wordpress.org/AJAX_in_Plugins
}
```

example.js

```
jQuery(document).on('click touch tap', '.example-selector', function(){
    jQuery.ajax({
        type: "POST",
        url: ajax.url,
        data: {
            nonce: ajax.nonce,
            action: 'example_function',
            data: {
                firstname: 'John',
                lastname: 'Doe'
            },
        },
        success: function(response){
            //Success
        },
        error: function(XMLHttpRequest, textStatus, errorThrown){
            //Error
        },
        timeout: 60000
    });
```

```
    return false;
});
```

wp_ajax - core functionality + _wpnonce check

functions.php:

```
function rm_init_js() {
    wp_enqueue_script( 'custom-ajax-script', get_template_directory_uri() . '/js/ajax.js',
array( 'jquery', 'wp-util' ), '1.0', true );
    // pass custom variables to JS
    wp_localize_script( 'custom-ajax-script', 'BEJS', array(
        'action' => 'custom_action',
        'nonce'  => wp_create_nonce( 'test-nonce' )
    ) );
}

add_action( 'wp_enqueue_scripts', 'rm_init_js' );

function rm_ajax_handler() {
    check_ajax_referer( 'test-nonce' );

    extract( $_POST );
    $data = compact( 'first_name', 'last_name', 'email' );

    foreach ( $data as $name => $value ) {
        switch ( $name ) {
            case 'first_name':
            case 'last_name':
                $data[ $name ] = ucfirst( sanitize_user( $value ) );
                break;
            case 'email':
                $data[ $name ] = sanitize_email( $value );
                break;
        }
    }

    $userID = email_exists( $data['email'] );

    if ( ! $userID ) {
        wp_send_json_error( sprintf( __( 'Something went wrong! %s try again!', 'textdomain'
), $data['first_name'] . ' ' . $data['last_name'] ) );
    }

    wp_update_user( array(
        'ID'           => $userID,
        'display_name' => $data['first_name'] . ' ' . $data['last_name'],
        'first_name'   => $data['first_name'],
        'last_name'    => $data['last_name'],
    ) );

    wp_send_json_success( sprintf( __( 'Welcome Back %s', 'textdomain' ), $data['first_name']
. ' ' . $data['last_name'] ) );
}

add_action( 'wp_ajax_custom_action', 'rm_ajax_handler' );
add_action( 'wp_ajax_nopriv_custom_action', 'rm_ajax_handler' );
```

ajax.js

```
;(function() {
    wp.ajax.post(BEJS.action, {
        first_name: 'john',
        last_name: '%65doe',
        email: 'john.doe@example.com',
        _ajax_nonce: BEJS.nonce
    }).done( function( response ) {
        alert(`Success: ${response}`);
    }).fail( function( response ) {
        alert(`Error: ${response}`);
    });
})();
```

OOP ajax submission using a simple class with nonce

You can copy and paste this whole plugin to try it. The class skeleton is used from here.

class-oop-ajax.cpp

```
<?php

/**
 * The plugin bootstrap file
 *
 * This file is read by WordPress to generate the plugin information in the plugin
 * Dashboard. This file defines a function that starts the plugin.
 *
 * @wordpress-plugin
 * Plugin Name:         Oop Ajax
 * Plugin URI:          http://
 * Description:         A simple example of using OOP principles to submit a form from the
front end.
 * Version:             1.0.0
 * Author:              Digvijay Naruka
 * Author URI:          http://
 * License:             GPL-2.0+
 * License URI:         http://www.gnu.org/licenses/gpl-2.0.txt
 * Text Domain:         oop-ajax
 * Domain Path:         /languages
 */

// If this file is called directly, abort.
if ( ! defined( 'WPINC' ) ) {
    die;
}

class Oop_Ajax {

  // Put all your add_action, add_shortcode, add_filter functions in __construct()
  // For the callback name, use this: array($this,'<function name>')
  // <function name> is the name of the function within this class, so need not be globally
```

```
unique
  // Some sample commonly used functions are included below
    public function __construct() {

        // Add Javascript and CSS for front-end display
        add_action('wp_enqueue_scripts', array($this,'enqueue'));

        // Add the shortcode for front-end form display
        add_action( 'init', array( $this, 'add_form_shortcode' ) );
        // Add ajax function that will receive the call back for logged in users
        add_action( 'wp_ajax_my_action', array( $this, 'my_action_callback') );
        // Add ajax function that will receive the call back for guest or not logged in users
        add_action( 'wp_ajax_nopriv_my_action', array( $this, 'my_action_callback') );

    }

    // This is an example of enqueuing a JavaScript file and a CSS file for use on the front
end display
    public function enqueue() {
        // Actual enqueues, note the files are in the js and css folders
        // For scripts, make sure you are including the relevant dependencies (jquery in this
case)
        wp_enqueue_script('my-ajax-script', plugins_url('js/oop-ajax.js', __FILE__),
array('jquery'), '1.0', true);

        // Sometimes you want to have access to data on the front end in your Javascript file
        // Getting that requires this call. Always go ahead and include ajaxurl. Any other
variables,
        // add to the array.
        // Then in the Javascript file, you can refer to it like this:
my_php_variables.ajaxurl
        wp_localize_script( 'my-ajax-script', 'my_php_variables', array(
            'ajaxurl' => admin_url('admin-ajax.php'),
            'nonce' => wp_create_nonce('_wpnonce')
        ));

    }

    /**
     * Registers the shortcode for the form.
     */
    public function add_form_shortcode() {

        add_shortcode( "oop-ajax-add-form", array( $this, "add_form_front_end" ) );

    }

    /**
     * Processes shortcode oop-ajax-add-form
     *
     * @param   array    $atts        The attributes from the shortcode
     *
     * @return    mixed    $output        Output of the buffer
     */
    function add_form_front_end($atts, $content) {

        echo "<form id='my_form'>";

            echo "<label for='name'>Name: </label>";
            echo "<input id='name' type='text' name='name' ";
```

```
            echo "<br>" ;

            echo "<label id='email' for='email'>Email: </label>" ;
            echo "<input type='text' name='email'>";

            echo "<br>" ;

            echo "<input type='hidden' name='action' value='my_action' >" ;
            echo "<input id='submit_btn' type='submit' name='submit' value='submit'> ";

        echo "</form><br><br>";
        echo "<div id='response'>ajax responce will be here</div> ";
    }

     /**
     * Callback function for the my_action used in the form.
     *
     * Processses the data recieved from the form, and you can do whatever you want with it.
     *
     * @return    echo   response string about the completion of the ajax call.
     */
    function my_action_callback() {
        // echo wp_die('<pre>' . print_r($_REQUEST) . "<pre>");

        check_ajax_referer( '_wpnonce', 'security');

        if( ! empty( $_POST )){

            if ( isset( $_POST['name'] ) ) {

                $name = sanitize_text_field( $_POST['name'] ) ;
            }

            if( isset( $_POST['email'] ) ) {

                $email = sanitize_text_field( $_POST['email'] ) ;
            }

            ////////////////////////////////////////
            // do stuff with values
            // example : validate and save in database
            //          process and output
            ////////////////////////////////////////

            $response = "Wow <strong style= 'color:red'>". $name . "!</style></strong> you
rock, you just made ajax work with oop.";
            //this will send data back to the js function:
            echo $response;

        } else {

            echo "Uh oh! It seems I didn't eat today";
        }

        wp_die(); // required to terminate the call so, otherwise wordpress initiates the
termination and outputs weird '0' at the end.

    }

}
//initialize our plugin
```

```
global $plugin;

// Create an instance of our class to kick off the whole thing
$plugin = new Oop_Ajax();
```

oop-ajax.js

Put the js file inside the js directory i.e oop-ajax/js/oop-ajax.js

```
(function($) {
    'use strict';

    $("#submit_btn").on('click', function() {
        // set the data
        var data = {
             action: 'my_action',
             security: my_php_variables.nonce,
             name: $("#name").val(),
             email: $("#email").val()
        }

        $.ajax({
            type: 'post',
            url: my_php_variables.ajaxurl,
            data: data,
            success: function(response) {
                //output the response on success
                $("#response").html(response);

            },
            error: function(err) {
                console.log(err);
            }
        });

        return false;
    });
})(jQuery);
```

Chapter 12: Alternating main loop (pre_get_posts filter)

Syntax

- add_action('pre_get_posts', 'callback_function_name');
- function callback_function_name($query) {}
- // for PHP 5.3.0 or above
- add_action('pre_get_posts', function($query){});

Parameters

Parameter	Details
$query	*(WP_Query)* Loop object

Remarks

If you are using PHP 5.3.0 or above, you can use closures (anonymous functions)

```
add_action( 'pre_get_posts', function( $query ) {
    if( !$query->is_main_query() || is_admin() ) return;

    // this code will run only if
    // - this query is main query
    // - and this is not admin screen
});
```

Examples

Even more specific loop targeting

Let's say we want to change *main loop*, only for specific taxonomy, or post type.

Targeting only main loop on `book` post type archive page.

```
add_action( 'pre_get_posts', 'my_callback_function' );

function my_callback_function( $query ) {
    if( !$query->is_main_query() || is_admin() ) return;
    if( !is_post_type_archive( 'book' ) ) return;

    // this code will run only if
    // - this query is main query
    // - and this is not admin screen
```

```
    // - and we are on 'book' post type archive page
}
```

You can also check for category, tag or custom taxonomy archive page using `is_category()`, `is_tag()` and `is_tax()`.

You can use any *conditional tag* available in WordPress.

Show posts from only one category

```
add_action( 'pre_get_posts', 'single_category' );

function single_category( $query ) {
    if( !$query->is_main_query() || is_admin() ) return;

    $query->set( 'cat', '1' );
    return;
}
```

Pre get posts filter basic usage

Sometimes you would like to change main WordPress query.

Filter `pre_get_posts` is the way to go.

For example using `pre_get_posts` you can tell *main loop* to show only 5 posts. Or to show posts only from one category, or excluding any category etc.

```
add_action( 'pre_get_posts', 'my_callback_function' );

function my_callback_function( $query ) {
    // here goes logic of your filter
}
```

As you can see, we are passing *main loop* query object into our callback function argument.

Important note here: **we are passing argument as a reference**. It means that we do not need to return query or set any globals to get it working. As `$query` is a reference to the main query object, all changes we make on our object are immediately reflected in the main loop object.

Exclude category from posts list edit share

```
add_action( 'pre_get_posts', 'single_category_exclude' );

function single_category_exclude( $query ) {
    if( !$query->is_main_query() || is_admin() ) return;

    $query->set( 'cat', '-1' );
    return;
}
```

Change posts_per_page for main loop

All we need to do is to use `set()` method of `$query` object.

It takes two arguments, first what we want to set and second what value to set.

```
add_action( 'pre_get_posts', 'change_posts_per_page' );

function change_posts_per_page( $query ) {
    if( !$query->is_main_query() || is_admin() ) return;

    $query->set( 'posts_per_page', 5 );
    return;
}
```

Targeting only main WordPress loop

WordPress is applying `pre_get_posts` filter to literally any loop it generates. It means that all changes we are making in our callback function are applied to all exiting loops.

Obviously it is not what we want in most scenarios.

In most cases we would like to target only *main loop*, and only for non-admin screens.

We can use `is_main_query()` method and `is_admin()` global function to check if we are in the right place.

```
add_action( 'pre_get_posts', 'my_callback_function' );

function my_callback_function( $query ) {
    if( !$query->is_main_query() || is_admin() ) return;

    // this code will run only if
    // - this query is main query
    // - and this is not admin screen
}
```

Chapter 13: Child Theme Basics

Syntax

- **template** — is the name of the main WordPress theme, the parent.
- **child-theme** — is the package which overrides the **template**.

Remarks

I've been advertising that the use of a child theme is always a good thing but there is always a *But ...*

In our Template overwriting example let's imagine that the author of a theme is adding his own improvements to the sidebar template and there will be a new one at

`/themes/template/sidebar.php`

```
<?php
/**
 * The template for the sidebar containing the main widget area
 *
 * @link https://developer.wordpress.org/themes/basics/template-files/#template-partials
 */

if ( is_active_sidebar( 'sidebar-1' )  ) : ?>
    <aside id="secondary" class="sidebar widget-area" role="complementary">
        <?php dynamic_sidebar( 'sidebar-1' ); ?>
    </aside><!-- .sidebar .widget-area -->
<?php endif; ?>
```

Now our website won't benefit from the new `role="complementary"` spec because our child theme is still overwriting the **template** with its own file at `/themes/child-theme/sidebar.php`

It is our duty as website maintainers to keep a track about what templates do we overwrite and, in the imminent case of an update, look carefully at the changelog so you update the child theme files if necessary.

Examples

2) The purpose

The child themes are meant to be a safe way to keep customizations of the main template without fearing to lose them on a theme update.

Basically, whenever you want to edit a file inside the active template from your website you have to ask yourself "**What is going to happen when I will update the theme?**

And the answer is simple: **You lose them because the update will bring an entirely new theme**

folder.

So let's look at a child theme as a folder with files which will overwrite the files with the same path in the parent theme. Now let's bring some real examples:

Definition and requirements

A child theme is identified by WordPress when there is a directory (for example `child-theme`) inside `/wp-content/themes/` with the following files:

- `style.css`

  This file must start with a comment template like this:

  ```
  /*
  Theme Name: Example Child
  Author: Me
  Author URI: https://example.com/
  Template: example
  Text Domain: example-child-theme
  Domain Path: /languages/
  */
  ```

  The most important things to consider here are:

  - The `Template` name must be exactly the folder name which holds the parent theme (aka the parent theme slug)
  - Name your child theme in such a way you can easily identify it in Dashboard (usually just append `Child` to the parent's name, like `Example Child`)

- `index.php`
- `functions.php`

3) Template overwriting

The most common usage of a child theme is to override template parts. For example, a sidebar, if we have a theme with the following file at

`/themes/template/sidebar.php`

```
<?php
/**
 * The sidebar containing the main widget area.
 *
 * @link https://developer.wordpress.org/themes/basics/template-files/#template-partials
 */

if ( ! is_active_sidebar( 'sidebar-1' ) ) {
    return;
}?>
<div id="sidebar" class="widget-area">
    <?php dynamic_sidebar( 'sidebar-1' ); ?>
```

```
</div>
```

We can definitely add our own file in child theme (with the specifications from the first example) with the following file

`/themes/child-theme/sidebar.php`

```
<?php
/**
 * The sidebar containing the main widget area.
 */

if ( ! is_active_sidebar( 'sidebar-1' ) ) {
    return;
}?>
<div id="my-sidebar" class="my-own-awesome-class widget-area">
    <div class="my-wrapper">
        <?php dynamic_sidebar( 'sidebar-1' ); ?>
    </div>
</div>
```

Now `my-own-awesome-class` is safe in **child theme** and it won't be removed at any theme update and WordPress will always choose a template from child themes when it does find one on the same path.

Assets replacement

Even if it is not a best practice, sometimes you need to replace assets like CSS or JS files or libraries.

Note that the WordPress template overwriting system doesn't work with anything else than `.php` files, so when we talk about assets we refer to registered assets

One example could be the replacement of jQuery library with your desired version. In our child theme `functions.php` file we need to add a function that removes the current `jQuery` version and add our own from CDN(remember is just an example).

```
/**
 * Dequeue the jQuery script and add our own version.
 *
 * Hooked to the wp_print_scripts action, with a late priority (100),
 * so that it is after the script was enqueued.
 */
function my_own_theme_scripts() {
    // remove the current version
    wp_dequeue_script( 'jquery' );
    // register my desired version
    wp_register_script( 'jquery', 'https://code.jquery.com/jquery-3.1.0.min.js', false,
'3.1.0' );
    // load my version, here or somewhere else
    wp_enqueue_script( 'jquery' );
}
add_action( 'wp_print_scripts', 'my_own_theme_scripts' );
```

Chapter 14: Create a Post Programmatically

Syntax

- wp_insert_post(array $args, bool $wp_error);

Parameters

Parameter	Description
$args (Array Required)	A Key Value Array of the below elements.
$wp_error (Boolean Optional)	Return a WP_Error in case of failure.

Remarks

Arguments

The next table shows you a list of elements that you can use inside of the first parameter (Array).

Parameter	Description
ID	(Int) The post ID. If equal to something other than 0, the post with that ID will be updated. Default 0.
post_author	(Int) The ID of the user who added the post. Default is the current user ID.
post_date	(String) The date of the post. Default is the current time.
post_date_gmt	(String) The date of the post in the GMT timezone. Default is the value of $post_date.
post_content	(Mixed) The post content. Default empty.
post_content_filtered	(String) The filtered post content. Default empty.
post_title	(String) The post title. Default empty.
post_category	(Array) Array of post category values.
post_excerpt (String) The post excerpt. Default empty.	

Parameter	Description
post_status	(String) The post status. Default draft.
post_type	(String) The post type. Default post.
comment_status	(String) Whether the post can accept comments. Accepts open or closed. Default is the value of default_comment_status option.
ping_status	(String) Whether the post can accept pings. Accepts open or closed. Default is the value of default_ping_status option.
post_password	(String) The password to access the post. Default empty.
post_name	(String) The post name or slug. Default is the sanitized post title when creating a new post.
to_ping	(String) Space or carriage return-separated list of URLs to ping. Default empty.
pinged	(String) Space or carriage return-separated list of URLs that have been pinged. Default empty.
post_modified	(String) The date when the post was last modified. Default is the current time.
post_modified_gmt	(String) The date when the post was last modified in the GMT timezone. Default is the current time.
post_parent	(Int) Set this for the post it belongs to, if any. Default 0.
menu_order	(Int) The order the post should be displayed in. Default 0.
post_mime_type	(String) The mime type of the post. Default empty.
guid	(String) Global Unique ID for referencing the post. Default empty.
tax_input	(Array) Array of taxonomy terms keyed by their taxonomy name. Default empty.
meta_input	(Array) Array of post meta values keyed by their post meta key. Default empty.

Avoid Duplicated Posts

When you execute this function, you could probably get a duplicated post, at least that happened to me. (You can check it into the Post WordPress Section)

I found a solution:

```
if( !get_page_by_title( $title, 'OBJECT', 'post' ) ){
    $my_post = array('post_title' => $title,
        'post_content' => 'Content',
        'tags_input' => $tags,
        'post_category' => array(2),
        'post_status' => 'publish'
    );

    $result = wp_insert_post( $my_post );
}
```

Explanation

Before you save a new post, validate if the new post already exists using the post title as a parameter, if there's not a post title, you can save your new post.

Check get_page_by_title's documentation here.

Examples

Introduction

Sometimes we have another editor somewhere else instead of TinyMCE (Wordpress Default Editor). That happen when we are creating our own Theme, Plugin or something specific; and we need to write and manipulate a type of post and save it into our WP Database.

So, if you are on that situation, you can use a Wordpress Function called:

```
wp_insert_post( array $args, bool $wp_error );
```

Create a Basic Post

```
$basic_post_args = array(
    'post_title' => 'My Basic Post',
    'post_content' => 'This is a basic content',
    'post_status' => 'publish',
    'post_author' => 1,
    'post_category' => array(8, 59)
);

wp_insert_post( $basic_post_args );
```

Create a Basic Page

```
$basic_page_args = array(
    'post_title' => 'My Basic Page',
    'post_content' => 'This is a basic content',
    'post_type' => 'page',
    'post_status' => 'publish',
    'post_author' => 1
);

wp_insert_post( $basic_page_args );
```

Chapter 15: Create Template for Custom Post Type

Examples

Creating a custom template for Custom Post type book

To create a template for the single posts of our custom post type, we need to create a file called single-**post_type_name**.php where **post_type_name** is the name of our custom post type.

For example, if our custom post type is called “Books”, we need to create a PHP file called single-**book**.php. Note that we used the singular name of our custom post type.

Copy the contents of the single.php from the themes folder and paste it into the new template and save it then the template would be applied for the custom post type individual page.

Custom Post Type Templates

Custom Post Type Archive:

To create an archive template for a custom post type you have to set the `has_archive` argument equal to `true` in your `register_post_type()` function. In the example below a custom post type is created for an Event post type.

```
add_action( 'init', 'create_events_post_type' );
function create_events_post_type() {
  register_post_type( 'event',
    array(
      'labels' => array(
        'name' => __( 'Events' ),
        'singular_name' => __( 'Event' )
      ),
      'public' => true,
      'has_archive' => true,
    )
  );
}
```

To create a template for new custom post types you will have to create a new template file. To create a template for the single post pages you would name it `single-{post_type}.php` and `archive-{post_type}.php` for the archive.

The filename for our archive template will be `archive-event.php` and for the event page it would be `single-event.php`. Both files should be in your themes root directory.

An example archive template would look like this. Pulled from the twentyseventeen theme.

```
<?php
/**
 * The template for displaying archive pages
 *
 * @link https://codex.wordpress.org/Template_Hierarchy
 *
 * @package WordPress
 * @subpackage Twenty_Seventeen
 * @since 1.0
 * @version 1.0
 */

get_header(); ?>

<div class="wrap">

    <?php if ( have_posts() ) : ?>
        <header class="page-header">
            <?php
                the_archive_title( '<h1 class="page-title">', '</h1>' );
                the_archive_description( '<div class="taxonomy-description">', '</div>' );
            ?>
        </header><!-- .page-header -->
    <?php endif; ?>

    <div id="primary" class="content-area">
        <main id="main" class="site-main" role="main">

        <?php
        if ( have_posts() ) : ?>
            <?php
            /* Start the Loop */
            while ( have_posts() ) : the_post();

                /*
                 * Include the Post-Format-specific template for the content.
                 * If you want to override this in a child theme, then include a file
                 * called content-___.php (where ___ is the Post Format name) and that will be
used instead.
                 */
                get_template_part( 'template-parts/post/content', get_post_format() );

            endwhile;

            the_posts_pagination( array(
                'prev_text' => twentyseventeen_get_svg( array( 'icon' => 'arrow-left' ) ) .
'<span class="screen-reader-text">' . __( 'Previous page', 'twentyseventeen' ) . '</span>',
                'next_text' => '<span class="screen-reader-text">' . __( 'Next page',
'twentyseventeen' ) . '</span>' . twentyseventeen_get_svg( array( 'icon' => 'arrow-right' ) ),
                'before_page_number' => '<span class="meta-nav screen-reader-text">' . __(
'Page', 'twentyseventeen' ) . ' </span>',
            ) );

        else :

            get_template_part( 'template-parts/post/content', 'none' );

        endif; ?>
```

```
        </main><!-- #main -->
    </div><!-- #primary -->
    <?php get_sidebar(); ?>
</div><!-- .wrap -->

<?php get_footer();
```

Custom Post Type Single template:

Here is an example of a single template. Pulled from the twentyseventeen theme.

```
<?php
/**
 * The template for displaying all single posts
 *
 * @link https://developer.wordpress.org/themes/basics/template-hierarchy/#single-post
 *
 * @package WordPress
 * @subpackage Twenty_Seventeen
 * @since 1.0
 * @version 1.0
 */

get_header(); ?>

<div class="wrap">
    <div id="primary" class="content-area">
        <main id="main" class="site-main" role="main">

            <?php
                /* Start the Loop */
                while ( have_posts() ) : the_post();

                    get_template_part( 'template-parts/post/content', get_post_format() );

                    // If comments are open or we have at least one comment, load up the
comment template.
                    if ( comments_open() || get_comments_number() ) :
                        comments_template();
                    endif;

                    the_post_navigation( array(
                        'prev_text' => '<span class="screen-reader-text">' . __( 'Previous
Post', 'twentyseventeen' ) . '</span><span aria-hidden="true" class="nav-subtitle">' . __(
'Previous', 'twentyseventeen' ) . '</span> <span class="nav-title"><span class="nav-title-
icon-wrapper">' . twentyseventeen_get_svg( array( 'icon' => 'arrow-left' ) ) .
'</span>%title</span>',
                        'next_text' => '<span class="screen-reader-text">' . __( 'Next Post',
'twentyseventeen' ) . '</span><span aria-hidden="true" class="nav-subtitle">' . __( 'Next',
'twentyseventeen' ) . '</span> <span class="nav-title">%title<span class="nav-title-icon-
wrapper">' . twentyseventeen_get_svg( array( 'icon' => 'arrow-right' ) ) . '</span></span>',
                    ) );

                endwhile; // End of the loop.
            ?>

        </main><!-- #main -->
```

```
    </div><!-- #primary -->
    <?php get_sidebar(); ?>
</div><!-- .wrap -->

<?php get_footer();
```

Both template examples are pulling in partials to display the inner content.

If your child/parent theme has a single/archive template you should use that code as boilerplate for your new templates.

Chapter 16: Creating a custom template

Examples

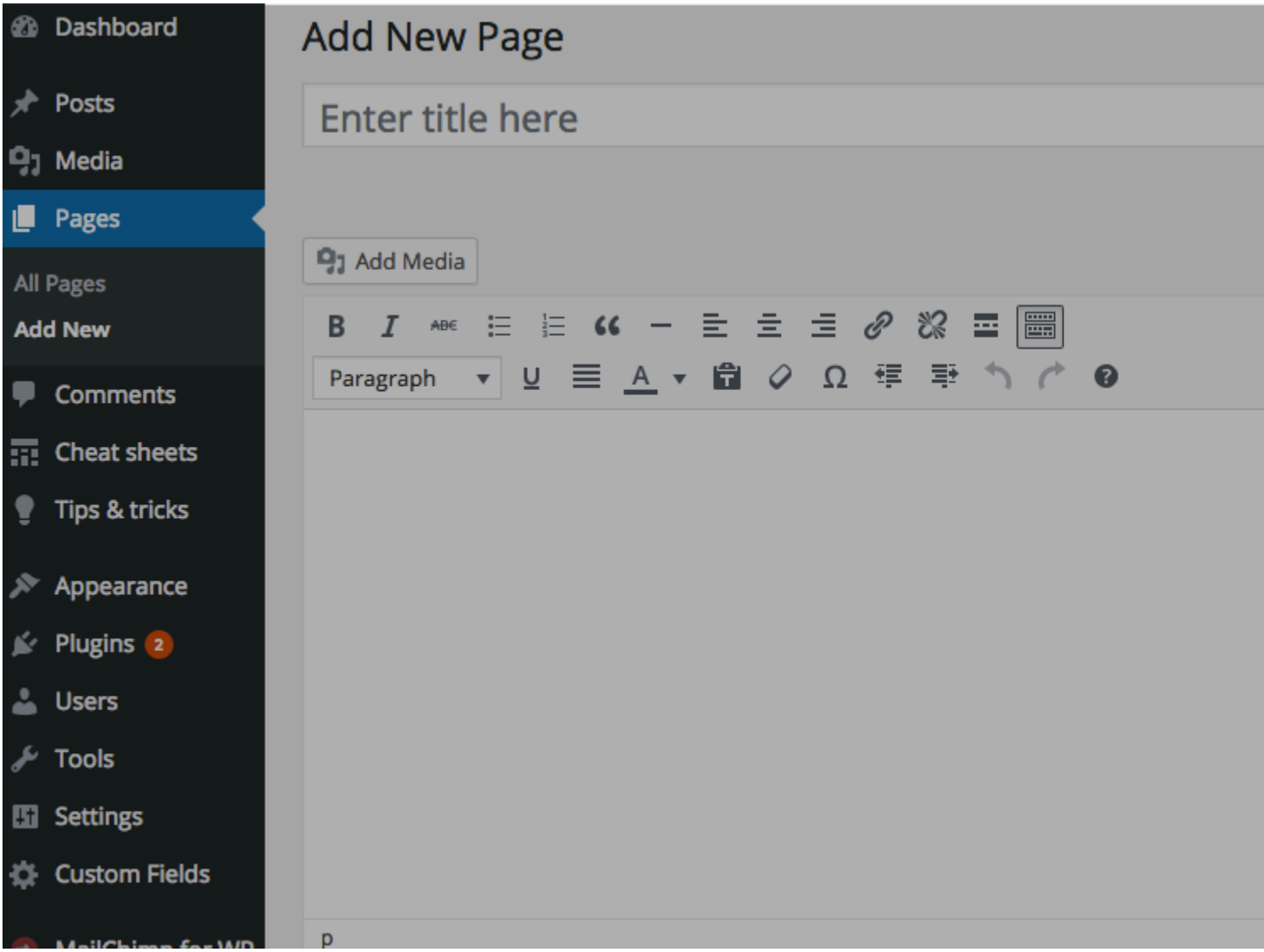
Creating basic blank template

To create a custom template we first need to create php file in a theme directory. You can name it almost any way you want. For this example we will create **example.php**

One and only thing we need to define inside our example.php, to be recognized by WordPress as a template, is template name. We do that buy putting special comment at the top of a file, like this:

```
<?php
/*
Template Name: Example
*/
?>
```

And now when we should see our template listed in **Template dropdown** in **Page Attributes Box**



Including header and footer in our template

Let's extend our template from above and include content from **header.php** and **footer.php**

Including header:

We will include header right after **Template name comment**

There are two common ways to do this. Both are right and work same, it's just about your style and how code looks

First way:

```
<?php
/*
Template Name: Example
*/
get_header();
?>
```

Second way:

```
<?php
/*
Template Name: Example
*/
?>
<?php get_header(); ?>
```

Including footer:

Including footer works the same way, there is only one thing we need to care about, and that is that we include footer after we included header. So the final template should look something like this.

```
<?php
/*
Template Name: Example
*/
get_header();
?>

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

Custom template with content

We will further extend our template and include title of the page and a content

```
<?php
/*
Template Name: Example
*/
```

```
get_header();

the_title();
the_content();

get_footer();
```

And if you want you can wrap them with HTML elements like this

```
<?php
/*
Template Name: Example
*/
get_header();

echo '<h1>' . the_title() . '</h1>';
echo '<section> . 'the_content() . '</section>';

get_footer();
```

Or if you prefer working like normal HTML file, without using echo

```
<?php
/*
Template Name: Example
*/
get_header();
?>

<h1><?php the_title(); ?></h1>
<section><?php the_content(); ?></section>

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

Chapter 17: Custom exerpts with excerpt_length and excerpt_more

Examples

Limit excerpt length to 50 words

Put the following code in **functions.php**:

```
function themify_custom_excerpt_length( $length ) {
   return 50;
}
add_filter( 'excerpt_length', 'themify_custom_excerpt_length', 999 );
```

Use 999 as the priority to ensure that the function runs after the default WordPress filter, otherwise it would override what is set here.

Adding a Read More link at the end of the excerpt

To do this, put the following code in **functions.php**:

```
function custom_excerpt_more($more) {
   return '<a href="'. get_permalink($post->ID) . '">Read More</a>';
}
add_filter('excerpt_more', 'custom_excerpt_more');
```

The results should look like this:

Adding a few dots at the end of the excerpt

In our **functions.php**

```
function new_excerpt_more( $more ) {
    return '.....';
}
add_filter('excerpt_more', 'new_excerpt_more');
```

We should get this:


Swift Srbija
Utisci sa prvog iOS meetupa
Mar 4, 2016
Utisci sa prvog iOS meetupa su fantastični, što bi u našem narodu rekli-prvo pa muško! Ovo okupljanje dokazalo je da iOS zajednica u Srbiji i te kako postoji i da uopšte nije mala kao što se mislilo. Na događaju je bilo nešto više od 100 ljudi, a predavanja su bila izuzetno zanimljiva i korisna. Organizatori.....
Ostavi komentar

Chapter 18: Custom Post Types

Syntax

- register_post_type($post_type, $args);

Parameters

Parameter	Details
$post_type	(string) (Required)
$args	(array/string) (Optional)

Examples

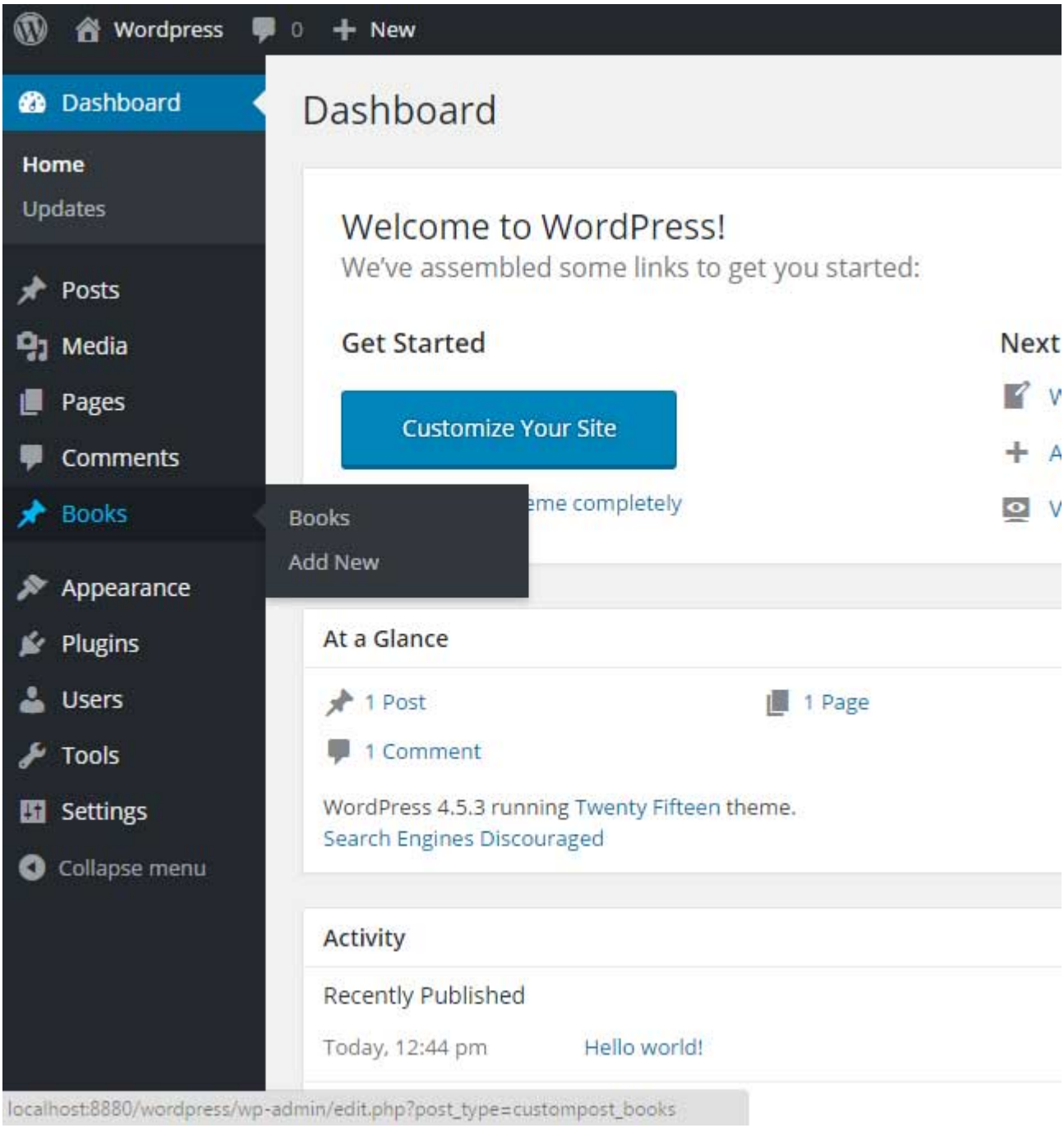
Registering a Custom Post Type

Say you have a library website, and you want to have a custom post type named *Books*. It can be registered as

```
function create_bookposttype() {
    $args = array(
        'public' => true,
        'labels' => array(
            'name' => __( 'Books' ),
            'singular_name' => __( 'Book' )
        ),
    );
    register_post_type( 'custompost_books', $args );
}

add_action( 'init', 'create_bookposttype' );
```

and, as simple as that is, you now have a custom post type registered.



This snippet can be placed in your theme `functions.php` file, or within a plugin structure.

Add Custom Post Types to Main Query

Registering a custom post type does not mean it gets added to the main query automatically. You need to use `pre_get_posts` filter to add custom post types to main query.

```
// Show posts of 'post' and 'book' custom post types on home page
add_action( 'pre_get_posts', 'add_my_post_types_to_query' );

function add_my_post_types_to_query( $query ) {
  if ( is_home() && $query->is_main_query() )
    $query->set( 'post_type', array( 'post', 'book' ) );
  return $query;
}
```

Adding Custom Post Types to Main RSS Feed

Registering a custom post type does not mean it gets added to the main RSS feed automatically.You need to use `request` filter to add custom post types to main RSS feed.

```
// Add 'books' custom post types on main RSS feed
function add_book_post_types_to_rss($qv) {
    if (isset($qv['feed']) && !isset($qv['post_type']))
        $qv['post_type'] = array('post', 'books', );
    return $qv;
}
add_filter('request', 'add_book_post_types_to_rss');
```

Register Custom Post Type

```
if ( ! function_exists('products_post_type') ) {

function products_post_type() {

    $labels = array(
        'name'                  => _x( 'Products', 'Post Type General Name', 'text_domain' ),
        'singular_name'         => _x( 'Product', 'Post Type Singular Name', 'text_domain' ),
        'menu_name'             => __( 'Products', 'text_domain' ),
        'name_admin_bar'        => __( 'Product', 'text_domain' ),
        'archives'              => __( 'Item Archives', 'text_domain' ),
        'attributes'            => __( 'Item Attributes', 'text_domain' ),
        'parent_item_colon'     => __( 'Parent Product:', 'text_domain' ),
        'all_items'             => __( 'All Products', 'text_domain' ),
        'add_new_item'          => __( 'Add New Product', 'text_domain' ),
        'add_new'               => __( 'New Product', 'text_domain' ),
        'new_item'              => __( 'New Item', 'text_domain' ),
        'edit_item'             => __( 'Edit Product', 'text_domain' ),
        'update_item'           => __( 'Update Product', 'text_domain' ),
        'view_item'             => __( 'View Product', 'text_domain' ),
        'view_items'            => __( 'View Items', 'text_domain' ),
        'search_items'          => __( 'Search products', 'text_domain' ),
        'not_found'             => __( 'No products found', 'text_domain' ),
        'not_found_in_trash'    => __( 'No products found in Trash', 'text_domain' ),
        'featured_image'        => __( 'Featured Image', 'text_domain' ),
        'set_featured_image'    => __( 'Set featured image', 'text_domain' ),
        'remove_featured_image' => __( 'Remove featured image', 'text_domain' ),
        'use_featured_image'    => __( 'Use as featured image', 'text_domain' ),
        'insert_into_item'      => __( 'Insert into item', 'text_domain' ),
        'uploaded_to_this_item' => __( 'Uploaded to this item', 'text_domain' ),
        'items_list'            => __( 'Items list', 'text_domain' ),
        'items_list_navigation' => __( 'Items list navigation', 'text_domain' ),
        'filter_items_list'     => __( 'Filter items list', 'text_domain' ),
    );
    $args = array(
        'label'                 => __( 'Product', 'text_domain' ),
        'description'           => __( 'Product information pages.', 'text_domain' ),
        'labels'                => $labels,
        'supports'              => array( 'title', 'editor', 'excerpt', 'author', 'thumbnail',
'comments', 'custom-fields', ),
        'taxonomies'            => array( 'category', 'post_tag' ),
        'hierarchical'          => false,
        'public'                => true,
        'show_ui'               => true,
```

```
        'show_in_menu'          => true,
        'menu_position'         => 5,
        'menu_icon'             => 'dashicons-products',
        'show_in_admin_bar'     => true,
        'show_in_nav_menus'     => true,
        'can_export'            => true,
        'has_archive'           => true,
        'exclude_from_search'   => false,
        'publicly_queryable'    => true,
        'capability_type'       => 'page',
        'show_in_rest'          => true,
    );
    register_post_type( 'product', $args );

}
add_action( 'init', 'products_post_type', 0 );

}
```

Custom Post Type using Twenty Fifteen WordPress Theme

You can use any name for the function.

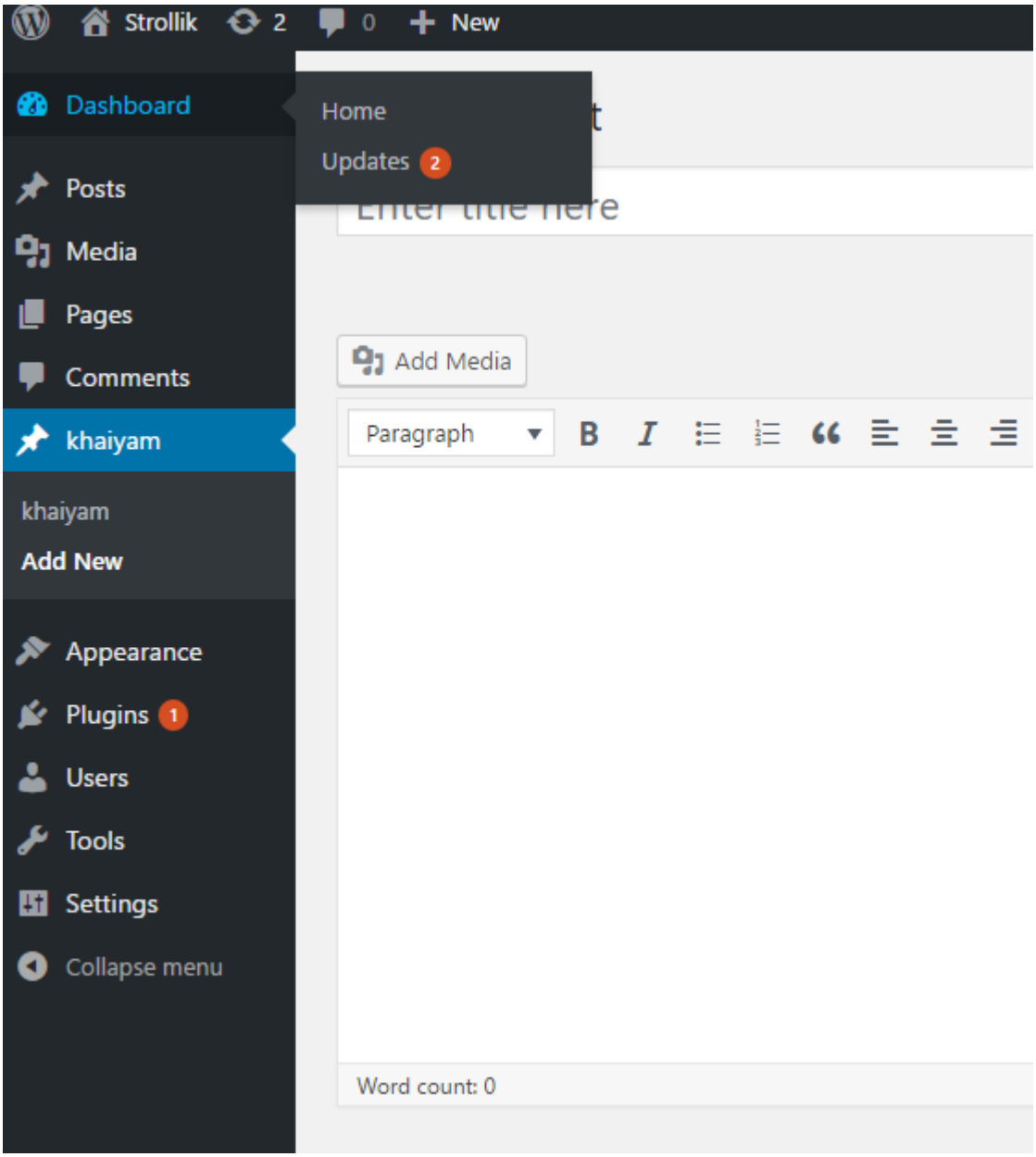
```
function custom_postype(){
    register_post_type('cus_post',array(

        'labels'=>array(
            'name'=>'khaiyam'// Use any name you want to show in menu for your users
            ),
        'public'=>true,// **Must required
        'supports'=>array('title','editor','thumbnail')// Features you want to provide on your
posts
        ));
}
add_action('after_setup_theme','custom_postytpe');
```

or

```
add_action('init','custom_postytpe');
```

You can use any of the hooks you want but of course they have different meaning and uses.



Custom post type in default search

You can add custom post type posts on default wordpress search, Add below code in theme functions.php

```
function my_search_filter($query) {
  if ( !is_admin() && $query->is_main_query() ) {
    if ($query->is_search) {
      $query->set('post_type', array( 'news','post','article' ) );
    }
  }
}
add_action('pre_get_posts','my_search_filter');
```

Chapter 19: Customizer Basics (Add Panel, Section, Setting, Control)

Examples

Add a Customizer Panel

```
<?php
/**
 * Panel: WPCustomize
 *
 * Basic Customizer panel with basic controls.
 *
 * @since    1.0.0
 * @package  WPC
 */

// Exit if accessed directly.
if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// Customize function.
if ( ! function_exists( 'wpc_panel_wpcustomize' ) ) {
    // Customize Register action.
    add_action( 'customize_register', 'wpc_panel_wpcustomize' );
    /**
     * Customize Panel.
     *
     * Adds a Panel, Section with basic controls.
     *
     * @param  object WP_Customize $wp_customize Instance of the WP_Customize_Manager class.
     * @since  1.0.0
     */
    function wpc_panel_wpcustomize( $wp_customize ) {
        // Panel: Basic.
        $wp_customize->add_panel( 'wpc_panel_wpcustomize', array(
            'priority'       => 10,
            'title'          => __( 'WPCustomize Panel Title', 'WPC' ),
            'description'    => __( 'Panel Description', 'WPC' ),
            'capability'     => 'edit_theme_options'
        ) );
    }
}
```

Add a Customizer Section With Basic Settings and their Controls

Panels can have sections, sections can have settings, and settings can have controls. Settings are saved in the database, while the controls for particular settings are only used to display their corresponding setting to the user.

This code creates a basic `section` in the `panel` from above. Inside are a few basic `settings` with `controls`

attached.

```
<?php
/**
 * Section: Basic
 *
 * Basic Customizer section with basic controls.
 *
 * @since     1.0.0
 * @package   WPC
 */

// Exit if accessed directly.
if ( ! defined( 'ABSPATH' ) ) {
    exit;
}

// Customize function.
if ( ! function_exists( 'wpc_customize_panel_basic' ) ) {
    // Customize Register action.
    add_action( 'customize_register', 'wpc_customize_panel_basic' );

    /**
     * Customize Panel.
     *
     * Adds a Panel, Section with basic controls.
     *
     * @param  object WP_Customize $wp_customize Instance of the WP_Customize_Manager class.
     * @since  1.0.0
     */
    function wpc_customize_panel_basic( $wp_customize ) {
        // Section: Basic.
        $wp_customize->add_section( 'wpc_section_basic', array(
            'priority'       => 10,
            'panel'          => 'wpc_panel_wpcustomize',
            'title'          => __( 'Basic Section Title', 'WPC' ),
            'description'    => __( 'Section Description.', 'WPC' ),
            'capability'     => 'edit_theme_options'
        ) );

        // Setting: Text.
        $wp_customize->add_setting( 'wpc_text', array(
            'type'                 => 'theme_mod',
            'default'              => 'Placeholder.',
            'transport'            => 'refresh', // Options: refresh or postMessage.
            'capability'           => 'edit_theme_options',
            'sanitize_callback'    => 'esc_attr'
        ) );

        // Control: Text.
        $wp_customize->add_control( 'wpc_text', array(
            'label'       => __( 'Text', 'WPC' ),
            'description' => __( 'Description', 'WPC' ),
            'section'     => 'wpc_section_basic',
            'type'        => 'text'
        ) );

        // Setting: Textarea.
        $wp_customize->add_setting( 'wpc_textarea', array(
            'type'                 => 'theme_mod',
            'default'              => 'Placeholder textarea.',
```

```
            'transport'           => 'refresh', // Options: refresh or postMessage.
            'capability'          => 'edit_theme_options',
            'sanitize_callback'   => 'exc_textarea'
        ) );

        // Control: Textarea.
        $wp_customize->add_control( 'wpc_textarea', array(
            'label'       => __( 'Textarea', 'WPC' ),
            'description' => __( 'Description', 'WPC' ),
            'section'     => 'wpc_section_basic',
            'type'        => 'textarea'
        ) );

        // Setting: Checkbox.
        $wp_customize->add_setting( 'wpc_checkbox', array(
            'type'                => 'theme_mod',
            'default'             => 'enable',
            'transport'           => 'refresh', // Options: refresh or postMessage.
            'capability'          => 'edit_theme_options',
            'sanitize_callback'   => 'wpc_sanitize_checkbox' // Custom function in
customizer-sanitization.php file.
        ) );

        // Control: Checkbox.
        $wp_customize->add_control( 'wpc_checkbox', array(
            'label'       => __( 'Checkbox', 'WPC' ),
            'description' => __( 'Description', 'WPC' ),
            'section'     => 'wpc_section_basic',
            'type'        => 'checkbox'
        ) );

        // Setting: Radio.
        $wp_customize->add_setting( 'wpc_radio', array(
            'type'                => 'theme_mod',
            'default'             => 'on',
            'transport'           => 'refresh', // Options: refresh or postMessage.
            'capability'          => 'edit_theme_options',
            'sanitize_callback'   => 'wpc_sanitize_select', // Custom function in customizer-
sanitization.php file.
        ) );

        // Control: Radio.
        $wp_customize->add_control( 'wpc_radio', array(
            'label'       => __( 'Radio', 'WPC' ),
            'description' => __( 'Description', 'WPC' ),
            'section'     => 'wpc_section_basic',
            'type'        => 'radio',
            'choices'  => array(
                'enable'  => 'Enable',
                'disable' => 'Disable'
            )
        ) );

        // Setting: Select.
        $wp_customize->add_setting( 'wpc_select', array(
            'type'                => 'theme_mod',
            'default'             => 'enable',
            'transport'           => 'refresh', // Options: refresh or postMessage.
            'capability'          => 'edit_theme_options',
            'sanitize_callback'   => 'wpc_sanitize_select' // Custom function in customizer-
sanitization.php file.
```

```
        ) );

        // Control: Select.
        $wp_customize->add_control( 'wpc_select', array(
            'label'       => __( 'Select', 'WPC' ),
            'description' => __( 'Description', 'WPC' ),
            'section'     => 'wpc_section_basic',
            'type'        => 'select',
            'choices'  => array(
                'enable'  => 'Enable',
                'disable' => 'Disable'
            )
        ) );
    }
}
```

Chapter 20: Customizer Hello World

Parameters

Parameter	Details
mytheme	A unique identifier for your theme (or child theme). This can be your theme slug

Examples

Hello World Example

The fundamental concept of the customizer is that admins can live preview changes to their site, and then permanently add them.

The following can be copied and pasted into a theme's `functions.php` file to

- Add a customizer section called `My First Section`
- Add a customizer setting called `Hello World Color` allowing the admin to choose a color
- Add a css rule for `.hello-world` that will corespond with the color chosen and default to `#000000` if nothing is chosen. The setting will be put in a `<style>` tag at the end of the `<head>`.

```
function mytheme_customize_register( $wp_customize ) {

    $wp_customize->add_section( 'my_first_section_id' , array(
        'title'      => __( 'My First Section', 'mytheme' ),
        'priority'   => 30,
    ) );

    $wp_customize->add_setting( 'hello_world_color' , array(
        'default'     => '#000000',
        'transport'   => 'refresh',
    ) );

    $wp_customize->add_control( new WP_Customize_Color_Control( $wp_customize, 'link_color',
array(
        'label'        => __( 'Hello World Color', 'mytheme' ),
        'section'    => 'my_first_section_id',
        'settings'   => 'hello_world_color',
    ) ) );


}
add_action( 'customize_register', 'mytheme_customize_register' );


function mytheme_customize_css()
{
    ?>
    <style type="text/css">
```

```
        .hello-world { color: #<?php echo get_theme_mod('hello_world_color', '000000'); ?>; }
    </style>
    <?php
}
add_action( 'wp_head', 'mytheme_customize_css');
```

Chapter 21: Debugging

Introduction

https://codex.wordpress.org/Debugging_in_WordPress

Debugging PHP code is part of any project, but WordPress comes with specific debug systems designed to simplify the process as well as standardize code across the core, plugins and themes.

Remarks

Plugins for debugging in WordPress:

- https://wordpress.org/plugins/query-monitor/
- https://wordpress.org/plugins/debug-bar/
- https://wordpress.org/plugins/debug-bar-console/
- https://wordpress.org/plugins/kint-debugger/
- https://wordpress.org/plugins/rest-api-console/

Examples

WP_DEBUG

`WP_DEBUG` is a PHP constant (a permanent global variable) that can be used to trigger the "debug" mode throughout WordPress. It is assumed to be false by default and is usually set to true in the `wp-config.php` file on development copies of WordPress.

```
define( 'WP_DEBUG', true );
define( 'WP_DEBUG', false );
```

WP_DEBUG_LOG

`WP_DEBUG_LOG` is a companion to WP_DEBUG that causes all errors to also be saved to a debug.log log file inside the /wp-content/ directory. This is useful if you want to review all notices later or need to view notices generated off-screen (e.g. during an AJAX request or wp-cron run).

```
//enable
define( 'WP_DEBUG_LOG', true );

//disable
define( 'WP_DEBUG_LOG', false );
```

WP_DEBUG_DISPLAY

`WP_DEBUG_DISPLAY` is another companion to WP_DEBUG that controls whether debug messages are

shown inside the HTML of pages or not. The default is 'true' which shows errors and warnings as they are generated. Setting this to false will hide all errors. This should be used in conjunction with WP_DEBUG_LOG so that errors can be reviewed later. Note: for WP_DEBUG_DISPLAY to do anything, WP_DEBUG must be enabled (true).

```
//enable
define( 'WP_DEBUG_DISPLAY', true );

//disable
define( 'WP_DEBUG_DISPLAY', false );
```

SCRIPT_DEBUG

`SCRIPT_DEBUG` is a related constant that will force WordPress to use the "dev" versions of core CSS and JavaScript files rather than the minified versions that are normally loaded. This is useful when you are testing modifications to any built-in .js or .css files. Default is false.

```
//enable
define( 'SCRIPT_DEBUG', true );

//disable
define( 'SCRIPT_DEBUG', false );
```

SAVEQUERIES

The SAVEQUERIES definition saves the database queries to an array and that array can be displayed to help analyze those queries. The constant defined as true causes each query to be saved, how long that query took to execute, and what function called it. NOTE: This will have a performance impact on your site, so make sure to turn this off when you aren't debugging.

```
define( 'SAVEQUERIES', true );
```

The array is stored in the

```
global $wpdb->queries;
```

Example wp-config.php and good practices for Debugging

The following code, inserted in your wp-config.php file, will log all errors, notices, and warnings to a file called debug.log in the wp-content directory. It will also hide the errors so they do not interrupt page generation.

```
 // Enable WP_DEBUG mode
define( 'WP_DEBUG', true );

// Enable Debug logging to the /wp-content/debug.log file
define( 'WP_DEBUG_LOG', true );

// Disable display of errors and warnings
```

```
define( 'WP_DEBUG_DISPLAY', false );
@ini_set( 'display_errors', 0 );

// Use dev versions of core JS and CSS files (only needed if you are modifying these core
files)
define( 'SCRIPT_DEBUG', true );
```

Good practice If you want add custom messages to debug log add folowing code in your plugin or theme.

```
//Checking is function_exists
    if ( !function_exists( 'print_to_log' ) ) {
            //function writes a message to debug.log if debugging is turned on.
            function print_to_log( $message )
            {
                if ( true === WP_DEBUG ) {
                    if ( is_array( $message ) || is_object( $message ) ) {
                        error_log( print_r( $message, true ) );
                    } else {
                        error_log( $message );
                    }
                }
            }
        }
```

See logs in a separate file

When you have an ajax call, it's extremely difficult to get a log from inside of the callback function. But if you enable the debugging

```
define('WP_DEBUG', true);
```

and then after that add

```
ini_set('log_errors',TRUE);
ini_set('error_reporting', E_ALL);
ini_set('error_log', dirname(__FILE__) . '/error_log.txt');
```

you will have an `error.log.txt` file in your root folder where all your logs are located. you can even log them with

```
error_log( print_r( 'what I want to check goes here', true) );
```

inside your code. This will make your life a lot easier.

Chapter 22: Enqueuing scripts

Syntax

- wp_enqueue_script($handle, $src, $deps, $ver, $in_footer)

Parameters

Parameter	Details
$handle	*(string)* (Required) Name of the script. Should be unique.
$src	*(string)* (Optional) Full URL of the script, or path of the script relative to the WordPress root directory. *Default value: false*
$deps	*(array)* (Optional) An array of registered script handles this script depends on. *Default value: array()*
$ver	*(string | bool | null)* (Optional) String specifying script version number, if it has one, which is added to the URL as a query string for cache busting purposes. If version is set to false, a version number is automatically added equal to current installed WordPress version. If set to null, no version is added. *Default value: false*
$in_footer	*(bool)* (Optional) Whether to enqueue the script before `</body>` instead of in the `<head>`. *Default value: false*

Examples

Enqueuing scripts in functions.php

If you want to add `custom.js` script that is located in the `js/` folder of your theme, you'll need to enqueue it. In `functions.php` add

```
<?php

add_action( 'after_setup_theme', 'yourtheme_theme_setup' );

if ( ! function_exists( 'yourtheme_theme_setup' ) ) {
    function yourtheme_theme_setup() {

        add_action( 'wp_enqueue_scripts', 'yourtheme_scripts' );
        add_action( 'admin_enqueue_scripts', 'yourtheme_admin_scripts' );

    }
}
```

```
if ( ! function_exists( 'yourtheme_scripts' ) ) {
    function yourtheme_scripts() {

        wp_enqueue_script( 'yourtheme_custom', get_template_directory_uri().'/js/custom.js',
array( 'jquery' ), '1.0.0', true );

    }
}

if ( ! function_exists( 'yourtheme_admin_scripts' ) ) {
    function yourtheme_admin_scripts() {

        wp_enqueue_script( 'yourtheme_custom', get_template_directory_uri().'/js/custom.js',
array( 'jquery-ui-autocomplete', 'jquery' ), '1.0.0', true );

    }
}
```

Enqueue scripts for IE only

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

/**
 * Enqueue scripts (or styles) conditionally.
 *
 * Load scripts (or stylesheets) specifically for IE. IE10 and above does
 * not support conditional comments in standards mode.
 *
 * @link https://gist.github.com/wpscholar/4947518
 * @link https://msdn.microsoft.com/en-us/library/ms537512(v=vs.85).aspx
 */
function enqueue_my_styles_and_scripts() {

     // Internet Explorer HTML5 support
    wp_enqueue_script( 'html5shiv',get_template_directory_uri().'/js/html5shiv.js', array(),
'3.7.3', false);
    wp_script_add_data( 'html5shiv', 'conditional', 'lt IE 9' );

    // Internet Explorer 8 media query support
    wp_enqueue_script( 'respond', get_template_directory_uri().'/js/respond.js', array(),
'1.4.2', false);
    wp_script_add_data( 'respond', 'conditional', 'lt IE 9' );

}
```

Enqueuing Scripts conditionally for specific pages

You can use conditional operators in WordPress to enqueue scripts on specific pages of your Website.

```
function load_script_for_single_post(){
    if(is_single()){
        wp_enqueue_script(
                'some',
                get_template_directory_uri().'/js/some.js',
                array('jquery),
                '1.0.0',
```

```
                false
        );

    }
}
add_action('wp_enqueue_scripts','load_script_for_single_post');
```

In the above example, if the current webpage is single post, script will be enqueued. Otherwise *wp_enqueue_script* function will not be executed.

Chapter 23: Enqueuing Styles

Syntax

1. wp_enqueue_style($handle, $src, $dependency, $version, $media);

Parameters

Parameter	Details
`$handle`	(String) (Required) Unique name for the stylesheet.
`$src`	(String) (Optional) URL of stylesheet which will be used inside **link** tag's src attribute.
`$deps`	(String) (Optional) An array of stylesheet handles this stylesheet depends on.
`$ver`	(String) (Optional) String specifying stylesheet version of stylesheet.
`$media`	(String) (Optional) The media for which this stylesheet is created. e.g 'all', 'print', 'screen' etc

Examples

Including internal css file with another css file as a dependency

```
function themeSlug_enqueue_scripts() {
    wp_enqueue_style( 'themeSlug-reset', get_template_directory_uri() .'/css/reset.css',
'1.0.0' );
    wp_enqueue_style( 'themeSlug-style', get_template_directory_uri() .'/style.css',
'themeSlug-reset', '1.0.0');
}
add_action('wp_enqueue_scripts', 'themeSlug_enqueue_scripts');
```

Including internal css file

In this case `style.css` is located in root of the theme's folder

```
function themeSlug_enqueue_scripts() {
    wp_enqueue_style( 'themeSlug-style', get_template_directory_uri() .'/style.css', '1.0.0');
}
add_action('wp_enqueue_scripts', 'themeSlug_enqueue_scripts');
```

Including external css file

In this example we want to include font awesome icon font

```
function themeSlug_enqueue_scripts() {
    wp_enqueue_style( 'font-awesome', '//cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.6.3/css/font-awesome.css');
}
add_action('wp_enqueue_scripts', 'themeSlug_enqueue_scripts');
```

Enqueue stylesheets for IE only

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

/**
 * Enqueue styles (or scripts) conditionally.
 *
 * Load stylesheets (or scripts) specifically for IE. IE10 and above does
 * not support conditional comments in standards mode.
 *
 * @link https://gist.github.com/wpscholar/4947518
 * @link https://msdn.microsoft.com/en-us/library/ms537512(v=vs.85).aspx
 */
function enqueue_my_styles_and_scripts() {

    // Internet Explorer specific stylesheet.
    wp_enqueue_style( 'themename-ie', get_stylesheet_directory_uri() . '/css/ie.css', array(
'twentyfifteen-style' ), '20141010' );
    wp_style_add_data( 'themename-ie', 'conditional', 'lte IE 9' );

    // Internet Explorer 7 specific stylesheet.
    wp_enqueue_style( 'themename-ie7', get_stylesheet_directory_uri() . '/css/ie7.css', array(
'twentyfifteen-style' ), '20141010' );
    wp_style_add_data( 'themename-ie7', 'conditional', 'lt IE 8' );

}
```

Including internal css file for your Plugin class

```
class My_Plugin() {
  function __construct() {
    add_action( 'wp_enqueue_scripts', array( $this, 'init_fe_assets' ) );
  }

  public function init_fe_assests() {
    wp_enqueue_style( 'my-plugin', plugin_dir_url( __FILE__ ) .
'assets/css/frontend/plugin.css', array(), '0.0.1', true );
  }
}

new My_Plugin();
```

Add Alternative Stylesheets

```
    <?php wp_enqueue_style('theme-five', get_template_directory_uri() .
'/path/to/additional/css');
wp_style_add_data('theme-five', 'alt', true);
wp_style_add_data('theme-five', 'title', __('theme-five.css', 'your-theme-name')); ?>
```

Chapter 24: Function : wp_trim_words()

Syntax

- `<?php $trimmed_text = wp_trim_words( $text, $num_words = 55, $more = null ); ?>`

Parameters

Parameter	Details
`$text`	(String) (Required) Text that will be shortened or trimmed.
`$num_words`	(Integer) (Required) Number of words to which text will be restricted.
`$more`	(String) (Optional) What to append if $text needs to be trimmed.

Examples

Trimming post content

This function shortens the text to a specified number of words and returns the shortened text.

```
<?php echo wp_trim_words( get_the_content(), 40, '...' ); ?>
```

In the above example we are passing the post content to the function. It will restrict the length of the content to 40 words and will trim rest of the words.

Chapter 25: Function: add_action()

Syntax

- add_action($tag, $function_to_add)
- add_action($tag, $function_to_add, $priority)
- add_action($tag, $function_to_add, $priority, $accepted_args)

Parameters

Parameter	Details
$tag	*(string) (Required)* The name of the action to which the `$function_to_add` is hooked
$function_to_add	*(callable) (Required)* The function which should be called when the action indicated by `$tag` is executed
$priority	*(int) (Optional) Default value:* `10` Used to specify the order in which the functions associated with a particular action are executed. Lower numbers correspond with earlier execution, and functions with the same priority are executed in the order in which they were added to the action.
$accepted_args	*(int) (Optional) Default value:* `1` The number of arguments the function accepts.

Remarks

The `add_action()` function creates an **Action Hook**, associating a PHP function with a particular *action* "tag" or name. When the action is "triggered" by a call to `do_action()` (or `do_action_ref_array()`) with a specific tag, all functions "hooked" to that tag will be executed.

In most cases, this function should be used in a theme's `functions.php` file or a plugin file - or another source file loaded by either.

This function is a part of the **Plugin API**

Examples

Basic Action Hook

The most basic application of `add_action()` is to add custom code to be executed at a certain location in a WordPress installation's source-code - either using actions supplied by the **Core** of WordPress, or ones created by third-party code such as plugins and themes.

To add content to the `<head></head>` section of the site - say to a add `<link>` meta element to indicate where copyright information for the site can be found - `add_action()` can be used to attach a function that prints the appropriate markup to the `'wp_head'` action (which "triggers" when WordPress builds the `<head>` section):

```
function add_copyright_meta_link() {
  echo( '<link rel="copyright" href="' . get_home_url() . '/copyright">' );
}

add_action( 'wp_head', 'add_copyright_meta_link' );
```

Action Hook Priority

Any number of functions may be "hooked" to any given action. In some instances it is important for a hooked function to execute before or after others, which is where the third parameter to `add_action()`, `$priority` comes into play.

If the `$priority` argument is omitted, the function will be attached with the default priority of `10`. When the action is "triggered", the "hooked" functions will be called starting with those added with the smallest `$priority`, and progressing to the functions with the largest `$priority`. Any hooked functions that share the same priority will be called in the order that they were added (the order in which their respective `add_action()` calls were executed).

For instance, say a third-party plugin is using a function hooked to the `'template_redirect'` action in order to forward visitors to the `daily-deal` page to an affiliate link for an external e-commerce site, but you'd like the redirection to only occur for logged-in users. You would need to use your own `'template_redirect'` hook to send logged-out visitors to the sign-in page. After determining that the third-party plugin attaches it's function with the default `$piority` of `10`, you could hook your function with a priority of `9` to ensure that your logged-in check happens first:

```
function redirect_deal_visitors_to_login() {
  if( is_page( 'daily-deal' ) && !user_is_logged_in() ) {
    wp_redirect( wp_login_url() );
    exit();
  }
}

add_action( 'template_redirect', 'redirect_deal_visitors_to_login', 9 );
```

Hooking Class & Object Methods to Actions

PHP Classes are powerful tool for improving code organization and minimizing naming collisions. At some point or another, the question of how to create an action hook for a class method inevitably arises.

The `$function_to_add` argument is often shown as a string containing the function's name, however the data-type of the argument is actually a "callable", which for our purposes can be summed up as "a reference to a function or method".

There are a number of callable formats that can be used to reference methods on classes and

objects. In all cases however, the referenced method *must* be publicly visible. A method is public when it is either prefixed with the `public` keyword, or no visibility keyword at all (in which case the method defaults to public).

Object Method Action Hooks

Object methods are executed on a particular instance of a class.

```
class My_Class {
  // Constructor
  function My_Class() {
    // (Instantiation logic)
  }

  // Initialization function
  public function initialize() {
    // (Initialization logic)
  }
}
```

After instantiating the above class as follows,

```
$my_class_instance = new My_Class();
```

the `initialize()` method would normally be invoked on the object by calling `$my_class_instance->initialize();`. Hooking the method to the `'init'` WordPress action is done by passing an array containing a reference to the instance and a string containing the object method's name:

```
add_action( 'init', [ $my_class_instance, 'initialize' ] );
```

If `add_action()` is called within an object method, the `$this` pseudo-variable can also be used:

```
class My_Class {
  // Constructor
  function My_Class() {
    // (Instantiation logic)
    add_action( 'init', [ $this, 'initialize' ] );
  }

  // Initialization function
  public function initialize() {
    // (Initialization logic)
  }
}
```

Class Method Action Hooks

Class methods are executed statically on a class rather than any particular instance. Given the

following class,

```
class My_Class {
  // Initialization function
  public static function initialize() {
    // (Initialization logic)
  }
}
```

the `initialize()` method would normally be invoked by using the `::` scope-resolution operator, i.e. `My_Class::initialize();`. Hooking a static class method to a WordPress can be done in a couple different ways:

- Using an array composed of a string containing the class name, and a string containing the method name:

  ```
  add_action( 'init', [ 'My_Class', 'initialize' ] );
  ```

- Passing a string containing a full reference to the method, including the `::` operator:

  ```
  add_action( 'init', 'My_Class::initialize' );
  ```

- If `add_action()` is called within a static class method, the `self` keyword or the `__CLASS__` magic-constant can be used in place of the class name. Note that this is generally inadvisable as the values of these items become somewhat counter-intuitive in the case of class inheritance.

  ```
  class My_Class {
    // Setup function
    public static function setup_actions() {
      add_action( 'init', 'self::initialize' );
    }

    // Initialization function
    public static function initialize() {
      // (Initialization logic)
    }
  }
  ```

Chapter 26: get_bloginfo()

Introduction

Retrieves information about the current site.

Syntax

- get_bloginfo($show , $filter)

Parameters

Parameter	Details
$show	*(string)* The site setting information to retrieve.
$filter	*(string)* The specification on whether to return a filtered value or not.

Remarks

$show

Values	Description	Example
'name' (Default)	Site title	`'Matt Mullenweg'`
'description'	Site tagline	`'Just another WordPress site'`
'wpurl'	URL of the WordPress installation. Same as the `site_url()` function	`'http://example.com'` , `'http://localhost/wordpress'`
'url'	URL of the site. Same as the `home_url()` function	`'http://example.com'` , `'http://localhost/wordpress'`
'admin_email'	Email address of the main Administrator	`'matt@mullenweg.com'`
'charset'	Character encoding of the pages and feeds	`'UTF-8'`
'version'	Current version of the WordPress installation	`'4.5'`
'html_type'	content-type value of the	`'text/html'`

Values	Description	Example
	HTML	
'text_direction'	Text direction determined by the site's language	`'ltr'`
'language'	ISO 639-1 based language code	`'en-US'`
'stylesheet_url'	URL of the stylesheet of the activated theme. Value priority: **Child theme** » Parent theme.	`'http://example.com/wp-content/themes/twentysixteen/style.css'`
'stylesheet_directory'	Resource location of the activated theme. Value priority: **Child theme** » Parent theme.	`'http://example.com/wp-content/themes/twentysixteen'`
'template_url'	URL directory of the activated theme. Value priority: **Parent theme** » Child theme.	`'http://example.com/wp-content/themes/twentysixteen'`
'template_directory'	Same as `'template_url'`	
'pingback_url'	Pingback XML-RPC file	`'http://example/xmlrpc.php'`
'atom_url'	Atom feed URL	`'http://example/feed/atom/'`
'rdf_url'	RDF/RSS 1.0 feed URL	`'http://example/feed/rdf/'`
'rss_url'	RSS 0.92 feed URL	`'http://example/feed/rss/'`
'rss2_url'	RSS 2.0 feed URL	`'http://example/feed/'`
'comments_atom_url'	Comments Atom feed URL	`'http://example/comments/feed/atom/'`
'siteurl'	*(deprecated)* Use 'url' instead	
'home'	*(deprecated)* Use 'url' instead	

$filter

Values	Description	Example
'raw' (Default)	No filters will be applied	*raw data*
'display'	Filters will be applied to the return value if `$show` is neither `'url'`, `'directory'`, `'home'`	*filtered data*

Examples

Getting the site title

```
<?php echo get_bloginfo( 'name' ); ?>
```

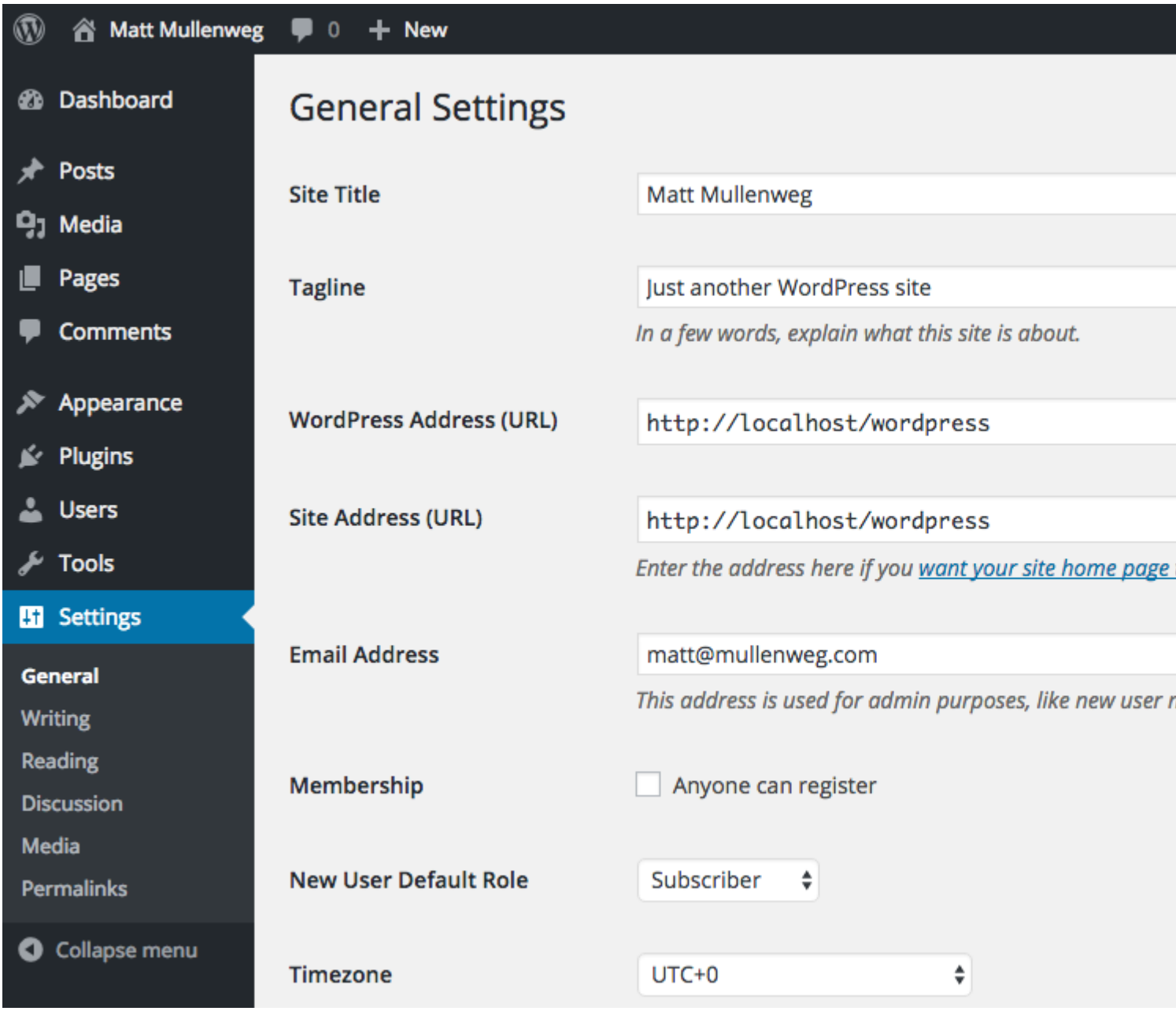
or

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

Output

```
Matt Mullenweg
```

Based on these sample settings



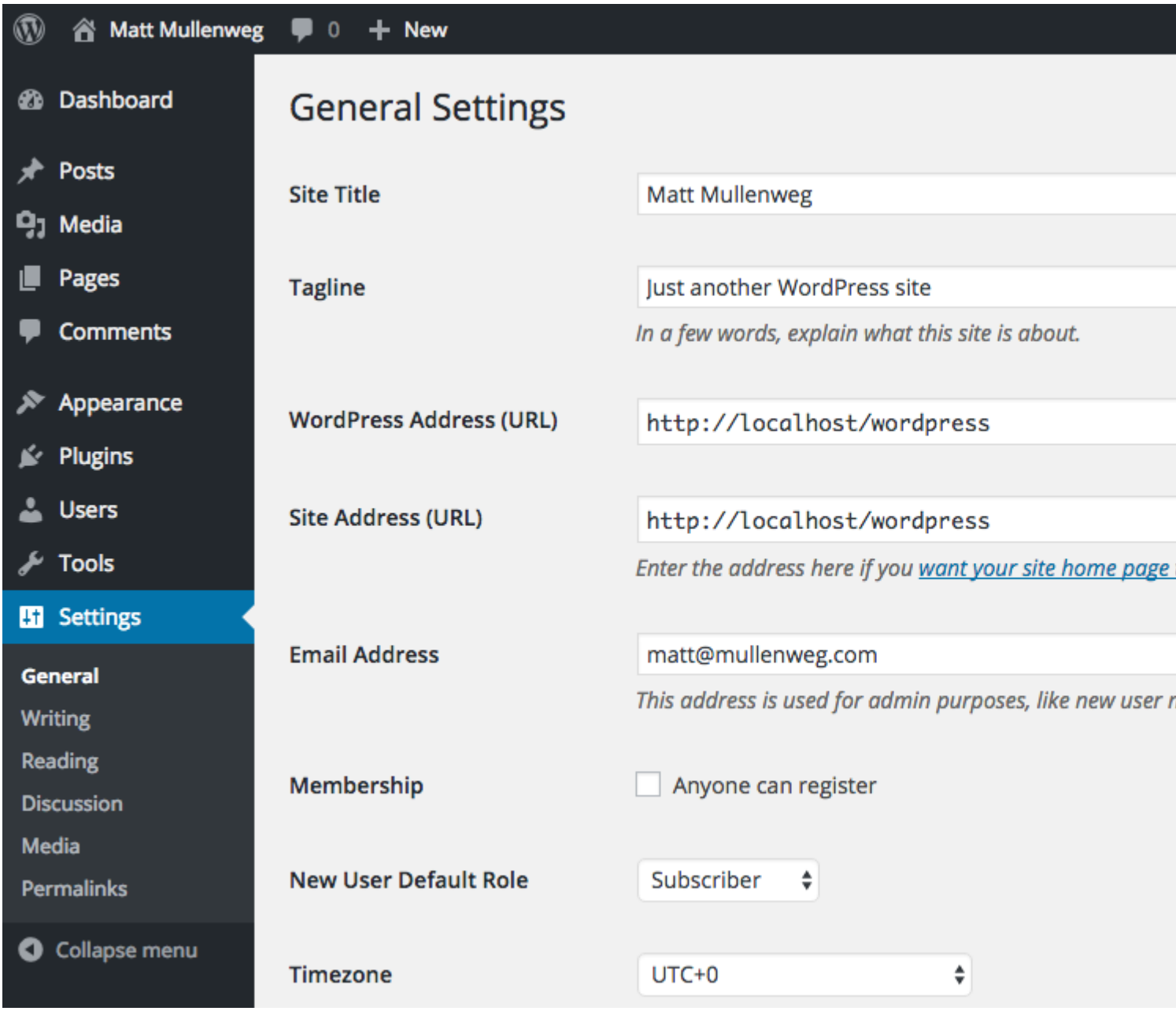
Getting the site tagline

```
<?php echo get_bloginfo( 'description' ); ?>
```

Output

```
Just another WordPress site
```

Based on these sample settings



Getting the active theme URL

```
<?php echo esc_url( get_bloginfo( 'stylesheet_directory' ) ); ?>
```

Output

```
http://example.com/wp-content/themes/twentysixteen
```

Alternatives

Internally, `get_bloginfo( 'stylesheet_directory' )` calls `get_stylesheet_directory_uri()`, so you may want to use that instead:

```
<?php echo esc_url( get_stylesheet_directory_uri() ); ?>
```

Many developers prefer to use these dedicated functions because of inconsistent naming conventions between them and `get_bloginfo()`. For example, `get_stylesheet_directory()` returns the child theme path; however, as our previous example illustrates, `get_bloginfo( 'stylesheet_directory' )` returns the child theme URL. If you use `get_stylesheet_directory_uri()` instead, there's less chance of confusion over whether you're retrieving a path or a URL.

Get site url

```
<?php echo esc_url(get_bloginfo('url')); ?>
```

or if you needed to link to a sub page

```
<?php echo esc_url(get_bloginfo('url') . '/some-sub-page');  ?>
```

Get Email Address of Site Administrator

We can use the `get_bloginfo` function to retrieve the email address of the site administrator.

```
<?php echo get_bloginfo('admin_email'); ?>
```

Chapter 27: get_home_path()

Introduction

Get the absolute filesystem path to the root of the WordPress installation.

Parameters

Parameter	Details
None	This function does not accept any parameters.

Remarks

Important difference between `get_home_path()` and `ABSTPATH`

Please keep in mind the difference between `ABSPATH` and `get_home_path()` if you have WordPress installed in a subfolder.

The `get_home_path()` function will always return a path **without** the subfolder:

- http://www.example.com - /var/www/htdocs/example
- http://www.example.com/wp - **/var/www/htdocs/example**

This is how it differs from `ABSPATH`, which will return different values:

- http://www.example.com - /var/www/htdocs/example
- http://www.example.com/wp - **/var/www/htdocs/example/wp**

`ABSPATH` is first defined in `wp-load.php` which will be located at `/var/www/htdocs/example/wp/wp-load.php` hence this is where `ABSPATH` will take its definition from.

`get_home_path()` checks if the `site_url` and `home_url` differ, and removes the substring from the path. Otherwise it returns `ABSPATH` value:

```
function get_home_path() {
    $home    = set_url_scheme( get_option( 'home' ), 'http' );
    $siteurl = set_url_scheme( get_option( 'siteurl' ), 'http' );
    if ( ! empty( $home ) && 0 !== strcasecmp( $home, $siteurl ) ) {
        $wp_path_rel_to_home = str_ireplace( $home, '', $siteurl ); /* $siteurl - $home */
        $pos = strripos( str_replace( '\\', '/', $_SERVER['SCRIPT_FILENAME'] ),
trailingslashit( $wp_path_rel_to_home ) );
        $home_path = substr( $_SERVER['SCRIPT_FILENAME'], 0, $pos );
        $home_path = trailingslashit( $home_path );
    } else {
        $home_path = ABSPATH;
    }
```

```
    return str_replace( '\\', '/', $home_path );
}
```

Using it in your code

Calling `get_home_path()` must be done in a context where `wp-admin/includes/file.php` has already been included.

For example using `get_home_path()` within the `admin_init` hook is fine, but using it within the `init` is not and will result in a PHP fatal error:

```
Call to undefined function get_home_path()
```

This file only gets included from within the admin (dashboard) context, if you absolutely need it outside of this context you will need to include the file yourself before calling the function:

```
require_once(ABSPATH . 'wp-admin/includes/file.php');
```

Examples

Usage

```
$path = get_home_path();
```

Return value:

`string`

Full filesystem path to the root of the WordPress installation, even if it's installed in a subfolder.

Example:

`/var/www/htdocs/example`

Chapter 28: get_option()

Introduction

Retrieves an option value based on an option name.

Syntax

- get_option($option, $default)

Parameters

Parameter	Details
$option	(string) Name of option to retrieve. Expected to not be SQL-escaped.
$default	(mixed) (Optional) Default value to return if the option does not exist.

Remarks

List of arguments for $option

- 'admin_email'
- 'blogname'
- 'blogdescription'
- 'blog_charset'
- 'date_format'
- 'default_category'
- 'home'
- 'siteurl'
- 'template'
- 'start_of_week'
- 'upload_path'
- 'users_can_register'
- 'posts_per_page'
- 'posts_per_rss'

Examples

Show blog title

Code

```
<h1><?php echo get_option( 'blogname' ); ?></h1>
```

Output

Name of the blog in H1 style

Show Character Set

Code

```
<p><?php echo esc_html( sprintf( __( 'Character set: %s', 'textdomain' ), get_option(
'blog_charset' ) ) ); ?></p>
```

Output

Character set: UTF-8

Handling non-existing options

Code

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

Output

false

Code

```
<?php echo get_option( 'something_bla_bla_bla', 'Oh no!' ); ?>
```

Output

Oh no!

Chapter 29: get_permalink()

Introduction

This function returns the full paralink of the current post or the designated post.

Syntax

- get_permalink($post, $leavename)

Parameters

Parameter	Details
$post	(int) (optional) Post ID or post object. Default is the the current post's id.
$leavename	(bool) (optional) Whether to keep post name or page name.

Remarks

For the parameter $leavename, it is false by default.

Examples

Simple use of get_parmalink

Code

```
echo get_permalink();
```

Output

The link of the current page, for example: http://website.com/category/name-of-post/

Specifying the post to get the link

Code

```
echo get_permalink( 44 );
```

Output

The link of the post id:44, for example: http://website.com/category/name-of-post/

Get the link without the post's name

Code

```
echo get_permalink( 44 , false );
```

Output

The link of the post id:44 without the name of the post, for example:
http://website.com/category/%postname%/

Chapter 30: get_template_part()

Syntax

- get_template_part($slug, $name);
- get_template_part($slug);

Parameters

Parameter	Details
$slug	*(string)* Generic template slug name
$name	*(string)* Specialized template name

Examples

Load a template part from a subfolder

```
get_template_part( 'foo/bar', 'page');
```

will require 'bar-page.php' from the directory 'foo'.

Get a specific file

You can get specific file using this function.

```
get_template_part('template-parts/layout');
```

Include layout.php file from template-parts subdirectory placed in the root of your theme folder.

Chapter 31: get_template_part()

Introduction

The purpose of this function is to standardize the way import partials or components of a theme into the main theme template. You could use a standard PHP SSI (server side includes), however, there are some benefits to using get_template_part(). Using this function reduces errors prone to less experienced developers trying to identify fully qualified path on the server.Also, it fails gracefully when files don't exist, and handles a custom hierarchy fallback system aka "fuzzy template searching".

Syntax

- get_template_part($slug)
- get_template_part($slug, $name)

Parameters

Parameter	Details
$slug	*(string)* The slug name of the custom template.
$name	*(string)* The name of the specialized template. Optional

Examples

Including a custom template

```
<?php get_template_part( 'foo' ); ?>
```

Includes

```
../wp-content/themes/your-theme-slug/foo.php
```

Including a custom template with a dash-separated filename

```
<?php get_template_part( 'foo','bar' ); ?>
```

Includes

```
../wp-content/themes/your-theme-slug/foo-bar.php
```

Including a custom template from inside a directory

```
<?php get_template_part( 'dir/foo' ); ?>
```

Includes

```
../wp-content/themes/your-theme-slug/dir/foo.php
```

Including a custom template with a dash-separated filename located inside a directory

```
<?php get_template_part( 'dir/foo', 'bar' ); ?>
```

Includes

```
../wp-content/themes/your-theme-slug/dir/foo-bar.php
```

Passing variable to custom template scope

```
<?php
set_query_var( 'passed_var', $my_var );
get_template_part( 'foo', 'bar' );
?>
```

Access it in `foo-bar.php`

```
<?php echo $passed_var; ?>
```

Chapter 32: get_template_part()

Syntax

- get_template_part('file-name-no-extension');

Parameters

Parameter	Description
file-name-no-extension	The name of the template part with no extension. E.g. 'foo' instead of 'foo.php'

Examples

Loading Template Part

Pulls the code from a certain specified file into another file where the call was made.

E.g. inside example.php

```
<h1>Hello World!</h1>
```

Inside page.php

```
// header code
get_template_part('example');
// rest of page code
```

Output:

```
// header code
<h1>Hello World</h1>
// rest of page code
```

Chapter 33: get_the_category()

Introduction

This function returns all categories as an array of the current post or page or the designated post or page.

Syntax

- get_the_category($id)

Parameters

Parameter	Details
$id	(int) (Optional) default to current post ID. The post ID.

Remarks

Please note that get_the_category() returns an array, which means that you can't directly echo the value returned.

Here is a list of objects of each category that you can print:

- term_id
- name
- slug
- term_group
- term_taxonomy_id
- taxonomy
- description
- parent
- count
- object_id
- filter
- cat_ID
- category_count
- category_description
- cat_name
- category_nicename
- category_parent

Examples

Get all names of categories of the post

Code

```
$categories = get_the_category();
 foreach( $categories as $category ) {
    echo  $category->name . '<br />';
}
```

Output

All names of categories of the current page, one on each line.

Get all ids of categories of the post

Code

```
$categories = get_the_category();
 foreach( $categories as $category ) {
    echo  $category->term_id . '<br />';
}
```

Output

All ids of categories of the current page, one on each line.

Chapter 34: get_the_title()

Introduction

This function returns the title of the current post or the designated post.

Syntax

- get_the_title($post)

Parameters

Parameter	Details
$post	(int) (optional) Post ID or post object. Default is the the current post's id.

Remarks

If you plan to get the title of a post or page using a post loop, it is suggested to use the_title() instead.

Examples

Simple use of get_the_title

Code

```
get_the_title();
```

Output

The title of the current post or page

Get the title of a specified post id

Code

```
get_the_title( 44 );
```

Output

The title of the post id:44.

Chapter 35: home_url()

Syntax

- home_url($path, $scheme);

Parameters

Parameter	Details
$path	(*String*,*Optional*) To adding more segment after the home url.
$scheme	(*String*,*Optional*) Scheme to give the home url context. Accepts 'http', 'https', 'relative', 'rest', or null.

Examples

Getting the Home URL

`home_url()` is used for retrieving the current site home url.

```
<?php echo esc_url( home_url( '/' ) ) ); ?>
```

Output

```
http://www.example.com
```

Site url

Returns the ‘site_url’ option with the appropriate protocol (https://)

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

Output

```
http://www.example.com
```

Chapter 36: How Can I integrate Markdown editor with Advance Custom Field's repeater Add-on.

Examples

Add MarkDown Editor

I found the solution. Please consider below mention steps.

Install wp Markdown Editor plugin.

Then Install "acf-wp-wysiwyg" for repeater field. Now you have to update in some files. Open this file and go to line number "180" or go to "create_field" function add

```
echo '<script> var simplemde = new SimpleMDE({element:
document.getElementById("'.$id.'")});jQuery(".quicktags-
toolbar").css("display","none");</script>';
```

Now under "acf-repeater" plugin open "input.js" file, line number "142"

replace

```
new_field_html = this.$el.find('> table > tbody > tr.row-clone').html().replace(/(=["]*[\w-
\[\]]*?)(acfcloneindex)/g, '$1' + new_id),
```

With

```
new_field_html = this.$el.find('> table > tbody > tr.row-clone').html().replace(/(["]*[\w-
\[\]]*?)(acfcloneindex)/g, '$1' + new_id),
```

Chapter 37: init

Syntax

1. add_action('init', callable $function)

Remarks

`init` is an action hook that gets fired after WordPress has finished loading but before any HTTP headers are sent.

Examples

Processing $_POST request data

add_action('init', 'process_post_data');

```
function process_post_data() {
    if( isset( $_POST ) ) {
        // process $_POST data here
    }
}
```

Processing $_GET request data

```
add_action('init', 'process_get_data');

function process_get_data() {
    if( isset( $_GET ) ) {
        // process $_GET data here
    }
}
```

Registering a custom post type

```
add_action( 'init', function() {
    register_post_type( 'event', array(
            'public' => true,
            'label'  => 'Events'
        );
    );
});
```

Registers a new custom post type with a label `Events` and a slug `event`

Chapter 38: Installation and Configuration

Examples

Wordpress on LAMP

I have tested the following steps on Amazon EC2 with Ubuntu 14.04

Here is a summary of command lines to install LAMP technology stack (before installing wordpress):

1: Install LAMP technology stack

Update linux

`#> sudo apt-get update` -> update package repositories information

`#> sudo apt-get upgrade` -> install package upgrades

Install apache2

```
#> sudo apt-get install apache2
```

Install php

```
#> sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

Install mysql

```
#> sudo apt-get install mysql-server php5-mysql
```

Install phpmyadmin

```
#> sudo apt-get install phpmyadmin
```

In case thing went wrong, remove installed package and its configuration file(s)

```
#> sudo apt-get purge package_name
```

Once the stack is installed here is the steps to complete wordpress installation:

2: Install WordPress

1. `wget https://wordpress.org/latest.zip`
2. `sudo mkdir /var/www/wordpress`
3. Move zip file into /var/www/wordpress and extract.
4. `cd /etc/apache2/sites-available/`
5. `sudo cp 000-default.conf wordpress.conf`

6. Modify wordpress.conf so apache2 knows to forward any request or your domain to the wordpress app folder.
7. `sudo a2ensite wordpress.conf`
8. `sudo service apache2 restart`
9. Go to PhpMyAdmin via your browser by youdomain.com/phpmyadmin . Create a new user “wordpress” and check to create the corresponding database. `cd /var/www/wordpress`
10. `sudo cp wp-config-example.php wp-config.php`
11. Modify config file to add your MySQL wordpress database information.
12. `sudo chown -R www-data:www-data /var/www/wordpress`
13. `sudo chmod -R 755 /var/www/wordpress/`
14. Open browser and type in your domain. You should see the WordPress installation page. Follow the instruction and you are done!

Installation WP in MAMP

It's quite simple to install wordpress in MAMP.

You have to follow a few simple steps:

1 - Download MAMP from here, it's free and you probably don't need the pro version.

2 - Install on your PC or Mac.

3 - Run the program -> you will see a little window open, from there you can set MAMP. At the moment leave all the pre-set values, for the first installation you don't need to complicate your life! In the future remember it's good practice to change your password and user name for the MySQL database. By default it's root.

4 - Click on "Open webStart page" - here you can see your data info like password, admin name and also info about MAMP.

5 - Click on tools -> phpMyAdmin and you will be redirected to the database page.

6 - Make a new database, click on new and give it the name you want, you will need this name later.

7 - Now look for a folder call "htdocs", it's inside the MAMP folder on your PC or Mac. If you use a Mac you need to one the application in Finder and open the app using "show package contents". Inside you will find the `htdocs` folder.

8 - Take the Wordpress folder and copy it inside the htdocs, you need put inside all folder, no put the zip file. Rename the folder, you can call with your project name.

9 - Comeback to the window of MAMP in your browser, and click on "My Site" - this will open a

URL call `http://localhost:8888` (8888 is the default port, you can change it if you want). Here you can see the folder you have put inside the `htdocs` folder, click on the name you have given to the folder.

10 - Now start a normal installation of WordPress, you need to use the Admin and Password. By default it's `root` and `root`, and use the name of the database you have created before.

11 - Run the installation and finish!

You will see your website on `http://localhost:8888/namefolder` Of course you need keep MAMP running.

Chapter 39: Making network requests with HTTP API

Syntax

- $response = wp_remote_get($url, $args);
- $response = wp_remote_post($url, $args);
- $response = wp_safe_remote_post($url, $args);

Parameters

Parameter	Details
$url	(string) (Required) Site URL to retrieve.
$args	(array) (Optional) Request arguments.

Remarks

Returns

(WP_Error | array) The response as an array, or WP_Error on failure.

Examples

GET a remote JSON resource

This snippet will grab a JSON formatted resource, decode it and print it in PHP array format.

```
// Fetch
$response = wp_remote_get( 'http://www.example.com/resource.json' );

if ( ! is_wp_error( $response ) ) {
  $headers = wp_remote_retrieve_headers( $response );

  if ( isset( $headers[ 'content-type' ] ) && 'application/json' === $headers[ 'content-type'
] ) {
    print_r( json_decode( wp_remote_retrieve_body( $response ) ) );
  }
}
```

Chapter 40: Meta Box

Introduction

Simple usage of a Meta Box in the wp-admin content editors

Syntax

- _x('Text', 'Description', 'textdomain') is used to add a description for the translation service instead of __('Text', 'textdomain') which is just the translation
- _ex('Text', 'Description', 'textdomain') is used to echo translated text with a description

Remarks

The content inside the render meta box can be anything. Instead of the values being directly integrated, you can also use an `include` with a PHP template and use `set_query_var` method to pass data to it. The save would work the same way.

Examples

A simple example with a regular input and a select input

```
/**
* Add meta box to post types.
*
* @since  1.0.0
*/
function myplugin_add_meta_box() {
    // Set up the default post types/
    $types = array(
        'post',
    );

    // Optional filter for adding the meta box to more types. Uncomment to use.
    // $types = apply_filters( 'myplugin_meta_box_types', $types );

    // Add the meta box to the page
    add_meta_box(
        'myplugin-meta-box', // Meta Box Id. Can be anything.
        _x( 'Custom Meta', 'Custom Meta Box', 'myplugin' ), // The title of the meta box.
Translation is optional. Can just be string.
        'myplugin_render_meta_box', // The render meta box function.
        $types, // Add the post types to which to add the meta box.
        'side', // Show on the side of edit.
        'high' // Show at top of edit.
    );
}

add_action( 'add_meta_boxes', 'myplugin_add_meta_box' );
```

```
/**
* Render the meta box.
*
* This shows examples of a basic input and a select inside a meta box. These can be anything.
*
* @since  1.0.0
*
* @param $post WP_Post The post being edited.
*/
function myplugin_render_meta_box( $post ) {
    // Get the current post meta values for our custom meta box.
    $city  = get_post_meta( $post->ID, 'city', true ); // True is for returning a single
value.
    $country = get_post_meta( $post->ID, 'country', true ); // True is for returning a single
value.

    // Add the WP Nonce field for security.
    wp_nonce_field( plugin_basename( __FILE__ ), 'myplugin_meta_nonce' );
?>

<p>
    <label for="city">
        <?php _ex( 'City', 'Custom Meta Box Template', 'myplugin' ); ?>
    </label>
    <input name="city" id="city" value="<?php echo $city; ?>" />
</p>
<p>
    <label for="country">
        <?php _ex( 'Country', 'Custom Meta Box Template', 'myplugin' ); ?>
    </label>
    <select name="country" id="country">
        <option value="United States" <?php selected( $country, 'United States' ); ?>><?php
_ex( 'United States', 'Custom Meta Box Template', 'myplugin' ); ?></option>
        <option value="Mexico" <?php selected( $country, 'Mexico' ); ?>><?php _ex( 'Mexico',
'Custom Meta Box Template', 'myplugin' ); ?></option>
        <option value="Canada" <?php selected( $country, 'Canada' ); ?>><?php _ex( 'Canada',
'Custom Meta Box Template', 'myplugin' ); ?></option>
    </select>
</p>

<?php
}

/**
* Save meta box data.
*
* @since  1.0.0
*
* @param $post_id int The Id of the Post being saved.
*/
function myplugin_save_meta_data( $post_id ) {
    // Verify this is not an auto save.
    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    // Validate the meta box nonce for security.
    if ( ! isset( $_POST['myplugin_meta_nonce'] ) || ! wp_verify_nonce(
$_POST['myplugin_meta_nonce'], plugin_basename( __FILE__ ) ) ) {
        return;
    }
```

```
    // Get the new values from the form.
    $city    = $_POST['city'];
    $country = $_POST['country'];

    // update_post_meta will add the value if it doesn't exist or update it if it does.
    update_post_meta( $post_id, 'city', $city );
    update_post_meta( $post_id, 'country', $country );

    /*
    * OPTIONAL ALTERNATIVE
    *
    * Instead of just using update_post_meta, you could also check the values and
    * issue create/update/delete on the post meta value.
    */
    // $current_city_value = get_post_meta( $post_id, 'city', true ); // True is for returning
a single value.
    //
    // // Add the post meta if it doesn't exist.
    // if ( $city && '' === $city ) {
    //     add_post_meta( $post_id, 'city', $city, true ); // True means the key is unique to
the post. False is default and more than one can be added.
    // }
    // // Edit the post meta if it does exist and there is a new value.
    // elseif ( $city && $city != $current_city_value ) {
    //     update_post_meta( $post_id, 'city', $city );
    // }
    // // Delete the post meta if there is no new value.
    // elseif ( '' === $city && $current_city_value ) {
    //     delete_post_meta( $post_id, 'city', $current_city_value ); // $current_city_value
is optional and is used to differentiate between other values with the same key.
    // }
}

add_action( 'save_post', 'myplugin_save_meta_data' );
```

Chapter 41: Options API

Introduction

Options are pieces of data that WordPress uses to store various preferences and configuration settings. The Options API is a simple and standardized way of storing data in the database. The API makes it easy to create, access, update, and delete options.

Syntax

- // Create new option within WordPress
  add_option($option, $value = , $deprecated = , $autoload = 'yes');
- // Removes an option from the database.
  delete_option($option);
- // Retrieve a saved option
  get_option($option, $default = false);
- // Update the value of an option that was already added.
  update_option($option, $newvalue);
- // There are also *_site_option() versions of these functions,
  // to manipulate network-wide options in WordPress Multisite
- // Create new network option
  add_site_option($option, $value = , $deprecated = , $autoload = 'yes');
- // Removes a network option
  delete_site_option($option);
- // Retrieve a saved network option
  get_site_option($option, $default = false);
- // Update the value of an option that was already added.
  update_site_option($option, $newvalue);

Remarks

The Options API is a simple and standardized way of working with data staored in the options table of MySQL database. The API makes it easy to create, read, update, and delete options.

Examples

get_option

get_option function is used to retrieve a value from from options table based on option name.

You can use the following code to get email address of a WordPress site administrator.

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

`get_option()` has an optional 2nd argument, which allows you to set a default value to return in the case that the requested option isn't set. By default, this argument is `false`.

To retrieve a text string, and use a boilerplate string if the text isn't set in the options table, you could do this:

```
<?php get_option( 'my_text', "I don't have anything written. Yet." ); ?>
```

add_option

add_option function ins used to insert new row into options table.

This will insert a new row in options table with option name *some_option_name* and value as *some_option_value*

```
<?php add_option( 'some_option_name', 'some_option_value' ); ?>
```

delete_option

delete_option function is used to delete an option from the options table.

This will delete *my_custom_option* from the options table.

```
<?php delete_option( 'my_custom_option' ); ?>
```

update_option

update_option function is used to update a value that already exists in the options table. If the option does not exist, then the option will be added with the option value.

This will set the default comment status to 'closed':

```
update_option( 'default_comment_status', 'closed' );
```

Chapter 42: Plugin development

Syntax

- add_action(string $tag, callable $function_to_add, int $priority = 10, int $accepted_args = 1)
- add_filter(string $tag, callable $function_to_add, int $priority = 10, int $accepted_args = 1)

Parameters

Parameter	Detail
$tag	(string) (Required) The name of the filter to hook the $function_to_add callback to.
$function_to_add	(callable) (Required) The callback to be run when the filter is applied.
$priority	(int) (Optional) Used to specify the order in which the functions associated with a particular action are executed. Lower numbers correspond with earlier execution, and functions with the same priority are executed in the order in which they were added to the action.Default value: 10
$accepted_args	(int) (Optional) The number of arguments the function accepts.Default value: 1

Remarks

The way Plugin hooks work is that at various times while WordPress is running, WordPress checks to see if any Plugins have registered functions to run at that time, and if so, the functions are run. These functions modify the default behavior of WordPress.

There are two kinds of hooks:

Filters give you the ability to change the value of a piece of data during the execution of WordPress. Callback functions for filters will be passed through a variable, modified, and then returned. They are meant to work in an isolated manner, and should never affect global variables or anything else outside of the function.

Actions, in contrast, allow you to add to or change how WordPress operates. Your callback function will run at a specific point in in the execution of WordPress, and can perform some kind of task, like echoing output to the user or inserting something into the database.

Filter Reference

Action Reference

HandBook

Plugin API

Filters vs Actions

Examples

Filter

```
add_filter('comment_text','before_comment');
add_filter('comment_text','after_comment');
function before_comment($comment_text){
            return 'input before comment'.$comment_text;
        }
function after_comment($comment_text){
            return $comment_text.'input after comment';
        }
```

Action

```
add_action('wp_head','hook_javascript');

function hook_javascript() {
    $output="<script> alert('Page is loading...'); </script>";
    echo $output;
}
```

Plugin development examples : Favorite Song Widget

```
    <?php
function wpshout_register_widgets() {
    register_widget( 'Favorite_Song_Widget');
}

add_action( 'widgets_init', 'wpshout_register_widgets' );

class Favorite_Song_Widget extends WP_Widget {

function Favorite_Song_Widget() {
    // Instantiate the parent object
    parent::__construct(
            'favorite_song_widget', // Base ID
            __('Favorite Song', 'text_domain'), // Name
            array( 'description' => __( 'Widget for playable favorite song', 'text_domain' ),
) // Args
    );
}

function widget( $args, $instance ) {
    echo $args['before_widget'];
    echo '<h3>Favorite Song Lists:</h3>';
    echo $instance['songinfo'];
    echo '<a href="' . $instance['link'] . '">Download it</a><br>';
```

```
    echo $instance['description'];
            echo $args['after_widget'];
}

function update($new_abc,$old_abc) {
    $instance = $old_abc;
    // Fields
    $instance['link'] = strip_tags($new_abc['link']);
    $instance['songinfo'] = strip_tags($new_abc['songinfo']);
            $instance['description'] = strip_tags($new_abc['description']);
    return $instance;
}

// Widget form creation
function form($instance) {
     $link = '';
    $songinfo = '';
            $description = '';
    // Check values
    if( $instance) {
        $link = esc_attr($instance['link']);
        $songinfo = esc_textarea($instance['songinfo']);
                    $description = esc_textarea($instance['description']);
    } ?>


    <p>
        <label for="<?php echo $this->get_field_id('link'); ?>"><?php _e('Link',
'wp_widget_plugin'); ?></label>
        <input class="widefat" id="<?php echo $this->get_field_id('link'); ?>" name="<?php
echo $this->get_field_name('link'); ?>" type="text" value="<?php echo $link; ?>" />
    </p>

    <p>
        <label for="<?php echo $this->get_field_id('songinfo'); ?>"><?php _e('Song Info:',
'wp_widget_plugin'); ?></label>
        <input class="widefat" id="<?php echo $this->get_field_id('songinfo'); ?>" name="<?php
echo $this->get_field_name('songinfo'); ?>" type="text" value="<?php echo $songinfo; ?>" />
    </p>

            <p>
        <label for="<?php echo $this->get_field_id('description'); ?>"><?php
_e('Description:', 'wp_widget_plugin'); ?></label>
        <textarea class="widefat" id="<?php echo $this->get_field_id('description'); ?>"
name="<?php echo $this->get_field_name('description'); ?>" type="text" value="<?php echo
$description; ?>"></textarea>
    </p>

            <p><a href="#" id="add-more-tabs"><?php _e('Add More Tabs', 'wp_widget_plugin');
?></a></p>

<?php }
```

}

Chapter 43: Post Formats

Remarks

The following Post Formats are available for users to choose from, if the theme enables support for them.

Note that while the actual post content entry won't change, the theme can use this user choice to display the post differently based on the format chosen. For example, a theme could leave off the display of the title for a "Status" post. How things are displayed is entirely up to the theme, but here are some general guidelines.

- aside - Typically styled without a title. Similar to a Facebook note update.
- gallery - A gallery of images. Post will likely contain a gallery shortcode and will have image attachments.
- link - A link to another site. Themes may wish to use the first tag in the post content as the external link for that post. An alternative approach could be if the post consists only of a URL, then that will be the URL and the title (post_title) will be the name attached to the anchor for it.
- image - A single image. The first tag in the post could be considered the image. Alternatively, if the post consists only of a URL, that will be the image URL and the title of the post (post_title) will be the title attribute for the image.
- quote - A quotation. Probably will contain a blockquote holding the quote content. Alternatively, the quote may be just the content, with the source/author being the title.
- status - A short status update, similar to a Twitter status update.
- video - A single video or video playlist. The first tag or object/embed in the post content could be considered the video. Alternatively, if the post consists only of a URL, that will be the video URL. May also contain the video as an attachment to the post, if video support is enabled on the blog (like via a plugin). audio - An audio file or playlist. Could be used for Podcasting.
- chat - A chat transcript

Examples

Adding post type to Theme

Add post-formats to post_type 'page'

```
add_post_type_support( 'page', 'post-formats' );
```

Next example registers custom post type 'my_custom_post_type', and add Post Formats.

Register custom post type 'my_custom_post_type'

```
add_action( 'init', 'create_my_post_type' );
function create_my_post_type() {
    register_post_type( 'my_custom_post_type',
      array(
        'labels' => array( 'name' => __( 'Products' ) ),
        'public' => true
    )
  );
}
```

Add post-formats to post_type 'my_custom_post_type'

```
add_post_type_support( 'my_custom_post_type', 'post-formats' );
```

Or in the function register_post_type(), add 'post-formats', in 'supports' parameter array. Next example is equivalent to above one.

Register custom post type 'my_custom_post_type' with 'supports' parameter

```
add_action( 'init', 'create_my_post_type' );
function create_my_post_type() {
    register_post_type( 'my_custom_post_type',
      array(
        'labels' => array( 'name' => __( 'Products' ) ),
        'public' => true,
        'supports' => array('title', 'editor', 'post-formats')
    )
  );
}
```

Add Theme Support for post

Function Call

```
add_theme_support( 'post-formats' )
```

Chapter 44: Querying posts

Syntax

- $the_query = new WP_Query($args);
- $posts_array = get_posts($args);

Parameters

Parameter	Description
$args	*(array)* An array of needed arguments for a query - can be custom tailored to your needs, e.g. querying posts from only one category, from custom post type or even querying certain taxonomy

Remarks

Query arguments are numerous. WP_Query() codex page has a list of parameters. Some of them are

- Author Parameters
- Category Parameters
- Tag Parameters
- Taxonomy Parameters
- Search Parameter
- Post & Page Parameters
- Password Parameters
- Type Parameters
- Status Parameters
- Pagination Parameters
- Order & Orderby Parameters
- Date Parameters
- Custom Field Parameters
- Permission Parameters
- Mime Type Parameters
- Caching Parameters
- Return Fields Parameter

One of the most important thing to have in mind is:

Never use query_posts()

`query_posts()` overrides the main query, and can cause problems in the rest of your theme. Any

time you need to modify the main query (or any query for that matter) is to use pre_get_posts filter. This will allow you to modify the query before it ran.

Also when you are querying posts, you should always reset it using wp_reset_postdata(). This will restore the global `$post` variable of the main query loop, and you won't have any issues later on (such as categories being excluded, because in your secondary loop you've excluded them and forgot to reset the query).

Examples

Using WP_Query() object

Creating a separate instance of the `WP_Query` object is easy:

```
$query_args = array(
                'post_type' => 'post',
                'post_per_page' => 10
            );

$my_query = new WP_Query($query_args);

if( $my_query->have_posts() ):
    while( $my_query->have_posts() ): $my_query->the_post();
       //My custom query loop
    endwhile;
endif;

wp_reset_postdata();
```

Notice that you need to build the query arguments array to your specification. For more details, look at WP_Query codex page.

Using get_posts()

`get_posts()` is a wrapper for a separate instance of a `WP_Query` object. The returned value is an array of post object.

```
global $post;

$args = array(
    'numberposts' => 5,
    'offset'=> 1,
    'category' => 1
);

$myposts = get_posts( $args );

foreach( $myposts as $post ) :
    setup_postdata($post); ?>
    <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>
<?php endforeach;
wp_reset_postdata(); ?>
```

Chapter 45: Remove Auto Line Breaks From Content and Excerpt

Introduction

For sites that rely on HTML by hand in the editor or excerpts, ones you want to code yourself, the auto line breaks can be an annoyance. You can disable them by removing these filters.

Remarks

These must be executed directly in an include file. Whether it is in functions.php or in another include file, these cannot be wrapped in a hook. They won't work on init or any other I have found so far.

They can also be included directly in a template like page.php to execute only for that template.

NOTE: DO NOT INCLUDE THIS IN A DISTRIBUTED THEME OR PLUGIN (unless it is disabled by default, like not including the include file it's in unless the user specifies).

This is bad practice to include in a site you don't control because it can and will break the output of any other themes or plugins.

Examples

Remove the Filters

```
// Remove the auto-paragraph and auto-line-break from the content
remove_filter( 'the_content', 'wpautop' );

// Remove the auto-paragraph and auto-line-break from the excerpt
remove_filter( 'the_excerpt', 'wpautop' );
```

Function to remove the filters

```
/**
 * Remove the automatic line breaks from content and excerpts.
 *
 * @since 1.0.0
 */
function remove_content_auto_line_breaks() {
    // Remove the auto-paragraph and auto-line-break from the content
    remove_filter( 'the_content', 'wpautop' );

    // Remove the auto-paragraph and auto-line-break from the excerpt
    remove_filter( 'the_excerpt', 'wpautop' );
}
```

```
// Execute the function
remove_content_auto_line_breaks();
```

Chapter 46: Remove Version from Wordpress and Stylesheets

Introduction

To make it more difficult for others to hack your website you can remove the WordPress version number from your site, your css and js. Without that number it's not possible to see if you run not the current version to exploit bugs from the older versions.

Additionally it can improve the loading speed of your site, because without query strings in the URL the css and js files can be cached.

Syntax

- add_filter($tag, $function_to_add, $priority, $accepted_args)

Parameters

Parameter	Details
$tag	*(string required)* Name of the filter where $function_to_add is hooked to
$function_to_add	*(callable required)* Name of the function which runs when the filter is applied
$priority	*(int optional)* place of $function_to_add between other functions in one action (default = 10)
$accepted_args	*(int optional)* Number of parameters which $function_to_add accepts (default = 1)

Remarks

Intended to improve site speed and safety.

Examples

Remove version numbers from css/js

Just add this function to your functions.php. It will remove the version from all enqueued js and css files.

```
function remove_cssjs_ver( $src ) {
   if( strpos( $src, '?ver=' ) )
   $src = remove_query_arg( 'ver', $src );
   return $src;
}

add_filter( 'style_loader_src', 'remove_cssjs_ver', 999 );
add_filter( 'script_loader_src', 'remove_cssjs_ver', 999 );
```

Remove version numbers from WordPress

If you add this to your functions.php it removes the WordPress version number from the RSS feed and the header.

```
function remove_wordpress_ver() {
     return '';
}
add_filter('the_generator', 'remove_wordpress_ver', 999);
```

Chapter 47: REST API

Introduction

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.

When you send content to or make a request to the API, the response will be returned in JSON. This enables developers to create, read and update WordPress content from client-side JavaScript or from external applications, even those written in languages beyond PHP.

Remarks

To get this WordPress REST API simple example to work for you, you need to learn how it works in more detail. Official documentation recommends learning about:

1. Routes/Endpoints - which are mappings of individual HTTP methods to routes known as "endpoints" - you do it using register_rest_route() function, and here you can find more about Routes and Endpoints.

2. Requests - WordPress REST API defines `WP_REST_Request` class which is used to store and retrieve information for the current request. `WP_REST_Request` objects are automatically generated for you whenever you make an HTTP request to a registered route. The data specified in the request will determine what response you get back out of the API. Here can learn more about the WP_REST_Request class.

3. Responses - are the data you get back from the API. The `WP_REST_Response` provides a way to interact with the response data returned by endpoints. In your endpoint definition you name the callback (response) function to serve your interaction. Here can learn more about the WP_REST_Response class.

4. Schema - Each endpoint requires and provides slightly different data structures, and those structures are defined in the API Schema. If you want maintainable, discoverable, and easily extensible endpoints it is recommended to use the schema. Here you can learn more about the Schema.

5. Controller Classes - they bring all elements together in a single place. With a controller class you can manage the registration of routes & endpoints, handle requests, utilize schema, and generate API responses. You have already learned about two controller classes: `WP_REST_Request` and `WP_REST_Response`. Here you can learn more about the Controller Classes

Note: Some of this information is taken from the official Wordpress REST APi Handbook

Examples

Complete working example

```
add_action('rest_api_init', 'my_rest_validate_endpoint' );
function my_rest_validate_endpoint() {

    // Declare our namespace
    $namespace = 'myrest/v1';

    // Register the route
    // Example URL matching this route:
    // http://yourdomain/wp-json/myrest/v1/guides/tag=europe/price=29
    register_rest_route($namespace,
                        // Using regular expressions we can initially validate the input
                        '/guides/tag=(?P<tag>[a-zA-Z0-9-]+)/price=(?P<price>[0-9]+)',
                        // We can have multiple endpoints for one route
                        array(
                            array(
                                'methods'   => 'GET',
                                'callback'  => 'my_get_guides_handler'
                            )
                        ),
                        // We can register our schema callback
                        // 'schema' => 'my_get_guide_schema',

        );

    // You can register another route here the same way

}


// The callback handler for the endpoint
function my_get_guides_handler(WP_REST_Request $request) {

    // Get the parameters:
    $tag = $request->get_param('tag');
    $price = $request->get_param('price');

    // Do something with the parameters
    // for instance: get matching guides from the DB into an array - $results
    // ...

    // Prepare the response
    $message = "We've found " . count($results) . " guides ";
    $message .= "(searching for a tag: " . $tag . ", with a price tag: " . $price . ")";

    // The response gets automatically converted into a JSON format
    return new WP_REST_Response(
        array(
            'results' => $results,
            'message' => $message,
            'status' => 'OK'
        ),
        200 );

}
```

Chapter 48: Run WordPress local with XAMPP

Introduction

With XAMPP you can install a web server on your local pc. It has an Apache web server, the database MariaDB (MySQL) and works with Perl and PHP. After the installation you are able to run and debug e.g. content management systems like WordPress on your local pc. You can use it with Windows, Linux, Mac OS X and Solaris.

Examples

1. Installing XAMPP

1. Download the installer for the current version of XAMPP
2. Go through the installation wizard and change nothing if you are not sure what you are changing.
3. After the installation you see the XAMPP control panel. Just click the start button of Apache and MySQL to run both with the basic configuration.
4. If your computer is connected to a local network, and you want access the web server from other computers too, be sure that your firewall allows it.

2. Setup a database after installing XAMPP

To run WordPress on your computer you need to configurate a database first. Be sure that Apache and MySQL are running (see Installing XAMPP step 3)

1. Start a web browser of your choice and enter "localhost" in the address.
2. Choose your preferred language and select in the category "Tools" -> `phpMyAdmin`.
3. Select the tab `Databases`.
4. Enter the name of your database (you will need this name later) below "Create database" and choose `utf8_general_ci` in the dropdown "Collation".
5. Click at your created database at the left side and select the tab `Users`.
6. Add a new user by clicking at `Add user`.
7. Enter your username, choose `local` in the "Host" dropdown (if you want access the database from other computers in the network choose `Any host`) and enter your password (you will need this later too).
8. Check if below "Database for user" `Grant all privileges on wildcard name` is checked.
9. Press `Go`.

3. Installing WordPress after setup the database

After you installed XAMPP and setup the MySQL database you can install WordPress.

1. Download the latest version of WordPress.
2. Unzip the file and insert the folder in the root directory of your webserver (if you used the suggested path during the setup of XAMPP it is "c:\xampp\htdocs").
3. All files which existing twice can be replaced.
4. Enter in the address field of your webbrowser "localhost/wordpress/wp-admin/install.php".
5. After choosing your language click `Continue` and then `Let's go`.
6. Replace all the default text fields with your chosen configuration (see "Setup a database"). Don't change "Database Host" and "Table Prefix", except you want to migrate your local WordPress installation online. Then keep in mind that security is important. You find more information in Set Prefix in New WordPress Installation.
7. Press `Submit` and than `Run the install`.
8. Now you have to enter the name of your site, an username, password and a valid e-mail.
9. Click `Install WordPress` to finish the setup and log into your new local WordPress site.

Chapter 49: Secure your installation

Remarks

WordPress websites are frequently hacked. This topic is for techniques and practices that increase the security of your WordPress installation beyond what is achieved in a base install.

Apart from this topic, another good place to read about securing a WordPress installation is the Hardening WordPress Codex page.

Examples

Disable File Editor

The file editor that ships with WordPress is a security risk. If an attacker gains admin access to your WordPress website they will be easily able to insert malicious code into theme and plugin files. It is also a risk with clients who don't know what they're doing. Once misplaced colon in the file editor can break a site and make it inaccessible from the browser.

In your WordPress `wp-config.php` file, disable the file editor by adding the following line of code.

```
define( 'DISALLOW_FILE_EDIT', true );
```

That line will have the desired effect when added to your theme's `functions.php` file too but it is better to add to `wp-config.php`.

If you are using WordPress CLI to install WordPress you can use the following command to create a `wp-config.php` file with file editing disabled.

```
/* declare variables beforehand or substitute strings in */
wp core config --dbname="$MYSQL_DBNAME" --dbuser="$MYSQL_USERNAME" --dbpass="$MYSQL_PASS" --
dbprefix="$WP_DBPREFIX"_ --locale=en_AU --extra-php <<PHP
define( 'DISALLOW_FILE_EDIT', true );
PHP
```

This method is useful if you install WordPress with a script.

Move wp-config.php

The most sensitive information of a WordPress install is stored in the `wp-config.php` file. If a hacker gets access to this file then they have total control of your website.

By default `wp-config.php` is stored in the WordPress install folder. To make this file harder to steal you can move it out of the web accessible folder. If you move it just one folder above, WordPress will automatically find it. If you move `wp-config.php` to a different location, create an empty file called `wp-config.php` in the WordPress installation folder. Then add the following:

```
define('ABSPATH', dirname(__FILE__) . '/');
// '../../wp-config.php' defines location two folders above installation folder.
// Substitute with actual location of wp-config.php file as necessary.
require_once(ABSPATH . '../../wp-config.php');
```

You may need to make `php` executable in the folder you place wp-config.php in. You should make php executable in as few folders as possible. A good system puts the WordPress install in `/path/to/wordpress/install/` and the config in `/path/to/wordpress/config`. You'd make sure the config folder is not web accessible and don't place any other sensitive information would be placed in `/path/to/` or higher in the folder hierarchy. In that case you'd write a line similar to the following in your `php.ini`:

```
open_basedir = "/path/to/wordpress/install/;/path/to/wordpress/config"
```

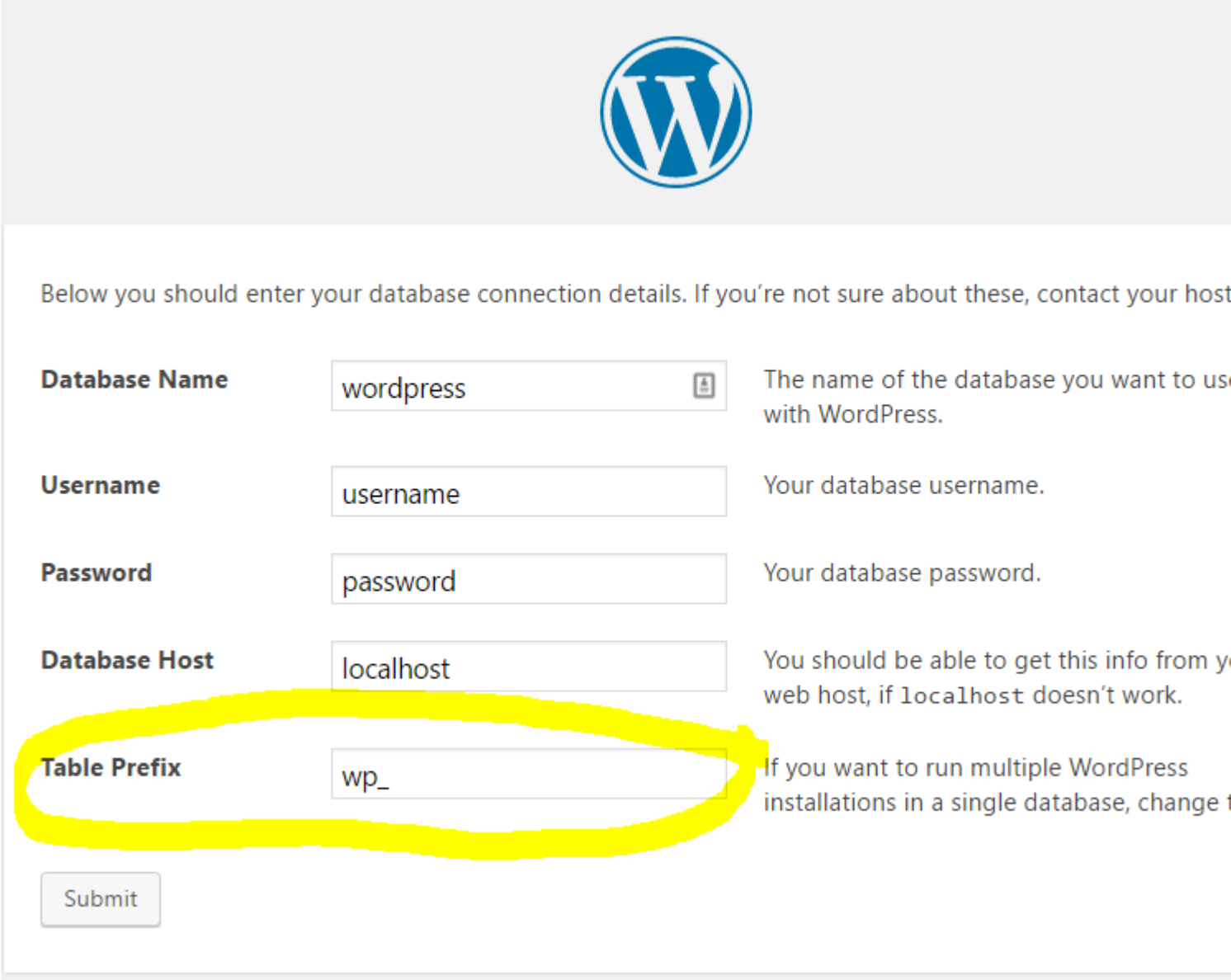
This technique is controversial and some people don't think it enhances security. Extensive discussion on the topic can be read at this WordPress StackExchange question.

Set a custom prefix for WordPress tables

When you install WordPress to your server, the installation script will place a prefix in front of all the WordPress MySQL table names. This prefix is set to 'wp_' by default. The WordPress posts table will be called `wp_posts` for example. By changing the table prefix you can create some security by obscurity. This way when a hacker attempts SQL injection attacks, they will have to guess the prefix of your table rather than just using 'wp_'. You can set this prefix to be whatever you like.

Set Prefix in New WordPress Installation

If using famous 5 minute installation Change prefix in field during installation.



If installing via WordPress CLI use the following command:

```
// set other variables above, or substitute your strings in.
WP_DBPREFIX=foo
wp core config --dbname="$MYSQL_DBNAME" --dbuser="$MYSQL_USERNAME" --dbpass="$MYSQL_PASS" --
dbprefix="$WP_DBPREFIX"_ --locale=en_AU
```

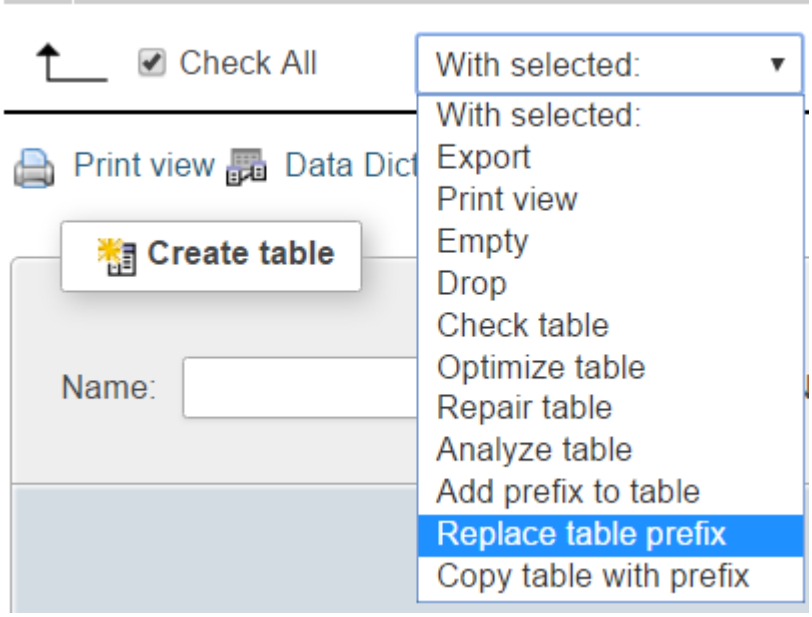
Change Prefix in Existing Installation

Changing the prefix is a little more difficult. Firstly use a FTP program like FileZilla to edit the `wp-config.php` file. Change the entry `$table_prefix = 'wp_';` to `$table_prefix = 'foo_';` substituting 'foo' for your desired prefix.

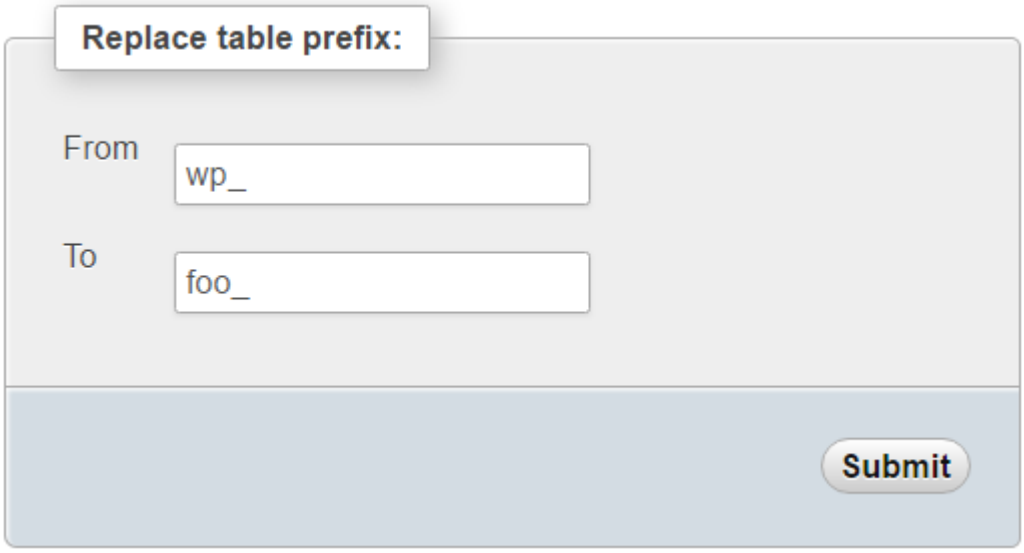
Next we'll need to edit the database. If you have access to phpMyAdmin, login and do the following:

- Select the WordPress database

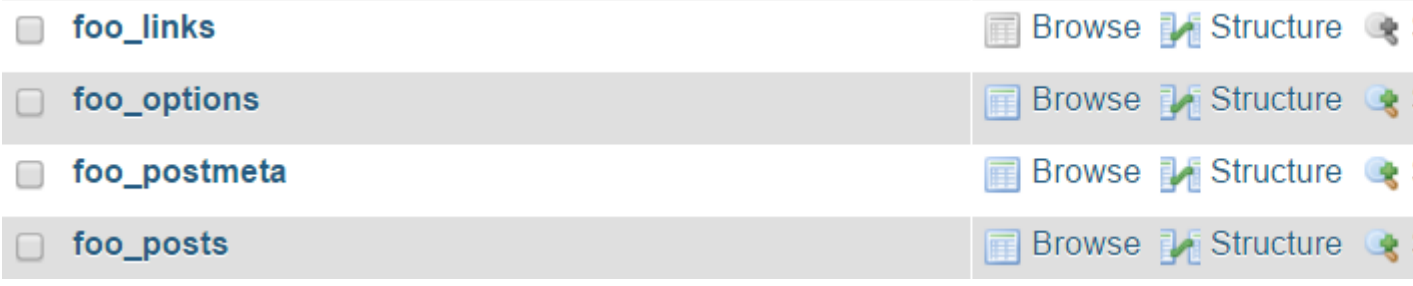
- Select all tables and in the dropdown select replace table prefix.



- In "From" type 'wp_'. In "To" type your prefix, 'foo_' in this example and press "Submit".



- Tables should now look like this:



If you can't use phpMyAdmin then use the following MySQL command:

```
RENAME table `wp_comments` TO `foo_comments`
```

You'll need to run that command for each table, substituting 'comments' for the other table names.

Next we need to change a few entries in some tables. Run this query on the 'foo_options' table

```
SELECT * FROM  foo_options WHERE option_name LIKE '%user_roles%'
```

A entry with option_name of 'wp_user_roles' should appear. In that entry change the 'option_name' entry from `wp_user_roles` to `foo_user_roles`.

Then open up 'foo_usermeta' table and find every entry with 'wp_' at the front.

	Edit Copy Delete	10	1	wp_capabilities	a:1:{s
	Edit Copy Delete	26	2	wp_capabilities	a:1:{s
	Edit Copy Delete	75	3	wp_capabilities	a:1:{s
	Edit Copy Delete	14	1	wp_dashboard_quick_press_last_post_id	2595
	Edit Copy Delete	29	2	wp_dashboard_quick_press_last_post_id	1283
	Edit Copy Delete	11	1	wp_user_level	10
	Edit Copy Delete	27	2	wp_user_level	10
	Edit Copy Delete	76	3	wp_user_level	0
	Edit Copy Delete	15	1	wp_user-settings	advlm
	Edit Copy Delete	41	2	wp_user-settings	editor
	Edit Copy Delete	16	1	wp_user-settings-time	14753
	Edit Copy Delete	42	2	wp_user-settings-time	14134

and change it to 'foo_'. The number of entries you must change will depend on how many users you have.

Edit Copy Delete	10	1	foo_capabilities	a:1
Edit Copy Delete	26	2	foo_capabilities	a:1
Edit Copy Delete	75	3	foo_capabilities	a:1
Edit Copy Delete	14	1	foo_dashboard_quick_press_last_post_id	25
Edit Copy Delete	29	2	foo_dashboard_quick_press_last_post_id	12
Edit Copy Delete	11	1	foo_user_level	10
Edit Copy Delete	27	2	foo_user_level	10
Edit Copy Delete	76	3	foo_user_level	0
Edit Copy Delete	15	1	foo_user-settings	ad
Edit Copy Delete	41	2	foo_user-settings	ed
Edit Copy Delete	16	1	foo_user-settings-time	14
Edit Copy Delete	42	2	foo_user-settings-time	14

That should be all you need to change the prefix in an existing installation

Chapter 50: Security in WordPress - Escaping

Syntax

- esc_html(string $text)
- esc_url(string $url, array $protocols, string $_context)
- esc_js(string $text)
- wp_json_encode(mixed $data, int $options, int $depth = 512)
- esc_attr(string $text)
- esc_textarea(string $text)

Remarks

Security should be always in mind when developing. Without security an app is open to various attacks such as SQL Injections, XSS, CSRF, RFI etc that can lead to serious problems.

Untrusted data comes from many sources (users, third party sites, your own database!, ...) and all of it needs to be validated both on input and output. (Source: WordPress Codex)

The data should be validated, sanitized or escaped depending the use and the purpose.

To validate is to ensure the data you've requested of the user matches what they've submitted. (Source: WordPress Codex)

Sanitization is a bit more liberal of an approach to accepting user data. We can fall back to using these methods when there's a range of acceptable input. (Source: WordPress Codex)

To escape is to take the data you may already have and help secure it prior to rendering it for the end user. (Source: WordPress Codex)

Examples

escape data in HTML code

esc_html should be used anytime we're outputting data inside HTML code.

```
<h4><?php echo esc_html( $title ); ?></h4>
```

escape a url

```
<a href="<?php echo esc_url( home_url( '/' ) ); ?>">Home</a>

<img src="<?php echo esc_url( $user_picture_url ); ?>" />
```

escape data in js code

`esc_js()` is intended to be used for inline JS, inside a tag attribute.

For data inside a `<script>` tag use `wp_json_encode()`.

```
<input type="text" onfocus="if( this.value == '<?php echo esc_js( $fields['input_text'] ); ?>'
) { this.value = ''; }" name="name">
```

`wp_json_encode()` encodes a variable into JSON, with some sanity checks.

Note that `wp_json_encode()` includes the string-delimiting quotes automatically.

```
<?php
$book = array(
    "title" => "JavaScript: The Definitive Guide",
    "author" => "Stack Overflow",
);
?>
<script type="text/javascript">
var book = <?php echo wp_json_encode($book) ?>;
/* var book = {
    "title": "Security in WordPress",
    "author" => "Stack Overflow",
}; */
</script>
```

or

```
<script type="text/javascript">
    var title = <?php echo wp_json_encode( $title ); ?>;
    var content = <?php echo wp_json_encode( $content ); ?>;
    var comment_count = <?php echo wp_json_encode( $comment_count ); ?>;
</script>
```

escape attributes

```
<input type="text" value="<?php echo esc_attr($_POST['username']); ?>" />
```

escape data in textarea

```
<textarea><?php echo esc_textarea( $text ); ?></textarea>
```

Chapter 51: Security in WordPress - Sanitization

Syntax

- sanitize_text_field(string $str)
- sanitize_title(string $title, string $fallback_title, string $context)
- sanitize_email(string $email)
- sanitize_html_class(string $class, string $fallback)
- sanitize_file_name(string $name)
- sanitize_user(string $username, boolean $strict)

Remarks

Security should be always in mind when developing. Without security an app is open to various attacks such as SQL Injections, XSS, CSRF, RFI etc that can lead to serious problems.

Untrusted data comes from many sources (users, third party sites, your own database!, ...) and all of it needs to be validated both on input and output. (Sourse: WordPress Codex)

The data should be validated, sanitized or escaped depending the use and the purpose.

To validate is to ensure the data you've requested of the user matches what they've submitted. (Sourse: WordPress Codex)

Sanitization is a bit more liberal of an approach to accepting user data. We can fall back to using these methods when there's a range of acceptable input. (Sourse: Wordpress Codex)

To escape is to take the data you may already have and help secure it prior to rendering it for the end user. (Sourse: WordPress Codex)

Examples

Sanitize text field

```
$title = sanitize_text_field( $_POST['title'] );
```

Sanitize title

The returned value is intended to be suitable for use in a URL, not as a human-readable title. Use sanitize_text_field instead.

```
$new_url = sanitize_title($title);
```

Sanitize email

```
$sanitized_email = sanitize_email('     admin@example.com!     ');
```

Sanitize html class

```
$post_class = sanitize_html_class( $post->post_title );
echo '<div class="' . $post_class . '">';
```

Sanitize file name

```
$incfile = sanitize_file_name($_REQUEST["file"]);
include($incfile . ".php");
```

Without sanitizing the file name an attacker could simple pass http://attacker_site/malicous_page as input and execute whatever code in your server.

Sanitize user name

```
$user = sanitize_user("attacker username<script>console.log(document.cookie)</script>");
```

$user value after sanitize is "attacker username"

Chapter 52: Shortcode

Examples

Registering shortcode

Shortcode is a small piece of code that can be added into the WordPress editor and will output something different once the page is published or previewed.

Frequently, shortcodes are added to the theme `functions.php` file, but that's not a good practice as shortcodes are expected to keep working after changing themes. Instead, write a plugin to add this functionality.

The structure for registering shortcode is:

```
function new_shortcode($atts, $content = null){
    // if parameters are needed in the shortcode
    // parameters can be set to default to something
    extract( shortcode_atts( array(
        'param_one' => 'h1'
    ), $atts ) );
    $shortcode = '<'.$param_one'>'.$content.'</'.$param_one.'>';
    return $shortcode;
}
// this is what registers the shortcode with wordpress
add_shortcode('demo-shortcode','new_shortcode');
```

Inside the WordPress editor, you can type:

```
[demo-shortcode param_one="h2"]Demo[/demo-shortcode]
// you don't need to insert param_one into the editor if it has a default value.
// having it in the editor will override the default
```

Once the page is published, this will turn into

```
<h2>Demo</h2>
```

Using Shortcodes in WordPress Backend

```
[footag foo="value of 1" attribute-2="value of 2"]
```

In wordpress admin we use pre defined shortcodes by writing the shortcode name inside square brackets and optionally adding attributes to it separating by space.

Adding New Shortcodes

```
function footag_func( $atts ) {
    return "foo = {$atts['foo']}";
```

```
}
add_shortcode( 'footag', 'footag_func' );
```

In plugins we can add shortcodes using the add_shortcode function.

The shortcode can be used in any Wordpress page or post just by enclosing it in square brackets.

```
[footag]
```

Using Shortcodes Inside PHP Code (themes and plugins)

```
<?php echo do_shortcode("[footag foo='Hi! I am a foo output']"); ?>
```

To print a shortcode using php use the `do_shortcode` function and echo the returned value.

Using Shortcodes in Widgets

```
add_filter( 'widget_text', 'shortcode_unautop' );
add_filter( 'widget_text', 'do_shortcode' );enter code here
```

Add this to a plugin or the `functions.php` file to enable shortcodes in widgets. The code first stops WordPress turning line breaks into paragraph tags and then lets shortcodes to parse for widgets. The order of the two lines is important.

Chapter 53: Shortcode with attribute

Syntax

- add_shortcode('your_short_code', 'your_function_name');

Parameters

Parameters	Discription and usage
$tag	(string) (required) Shortcode tag to be searched in post content Default: None
$func	(callable) (required) Hook to run when shortcode is found Default: None

Remarks

IMPORTANT – Don’t use camelCase or UPPER-CASE for your attributes

You can generate a shortcode with attribute Here

Examples

Examples of Shortcodes

WordPress shortcodes were introduced in 2.5

Here is come example of shortcode

```
[button]
```

to use shortcode direct into theme you have to use `do_shortcode()`

```
<?php echo do_shortcode('[button]'); ?>
```

To customize the button, we could simply add something like:

```
[button type="twitter"]
```

Or to make it even better, we could use an enclosing shortcode:

```
[button type="twitter"]Follow me on Twitter![/button]
```

Creating a self-closing shortcode

The simplest shortcode is the self-closing one. We're going to create a simple link to our Twitter account, and then add it in a blog post. All the code goes in `functions.php`, which is located in `/wp-content/themes/your-theme/`. If you don't have one, just create it and put the code in it.

```
<?php
function button_shortcode() {
return '<a href="http://twitter.com/rupomkhondaker" class="twitter-button">Follow me on
Twitter!</a>"';
}
add_shortcode('button', 'button_shortcode');
?>
```

Usage: `[button]`

Creating a self-closing shortcode with parameters

Creating a self-closing shortcode with parameters

```
<?php
function button_shortcode( $type ) {

    extract( shortcode_atts(
        array(
            'type' => 'value'
         ), $type ) );

    // check what type user entered
    switch ($type) {

        case 'twitter':
            return '<a href="http://twitter.com/rupomkhondaker" class="twitter-button">Follw
me on Twitter!</a>';
            break;

        case 'rss':
            return '<a href="http://example.com/rss" class="rss-button">Subscribe to the
feed!</a>'
            break;
    }
}
add_shortcode( 'button', 'button_shortcode' );
?>
```

Now you can choose what button to display by defining type in your shortcode.

```
[button type="twitter"]
[button type="rss"]
```

Creating an enclosing shortcode

enclosing shortcode

The enclosing shortcode allows you to embed content within your shortcode, just like BBCode if you've ever used that.

```
<?php
function button_shortcode( $attr, $content = null ) {
return '<a href="http://twitter.com/filipstefansson" class="twitter-button">' . $content .
'</a>';
}
add_shortcode('button', 'button_shortcode');
?>
```

To use this shortcode, you embedd the text you want to use like this:

```
[button]Follow me on Twitter![/button]
```

To make this button even better, we could add parameters just like we did in the previous example.

```
<?php
function button_shortcode( $atts, $content = null ) {
extract( shortcode_atts( array(
'account' => 'account',
 'style' => 'style'
 ), $atts ) );
return '<a href="http://twitter.com/' . esc_attr($account) . '" class="twitter-button ' .
esc_attr($style) . '">' . $content . '</a>';
}
add_shortcode('button', 'button_shortcode');
?>
```

Usage:

```
[button account="rupomkhondaker" style="simple"]Follow me on Twitter![/button]
```

Shortcodes in Widgets

By default, WordPress does not support shortcodes within Sidebar Widgets. It only expands the shortcodes within the content of a Post, Page, or custom post type. To add shortcode support to sidebar widgets, you can install a plugin, or use the below code:

```
add_filter( 'widget_text', 'shortcode_unautop' );
add_filter( 'widget_text', 'do_shortcode' );
```

It is important that these lines be added in this order. The first line prevents WordPress from turning line breaks into paragraph tags, since this keeps shortcodes from working. The second line is the one that makes the shortcodes work.

Chapter 54: Shortcodes

Examples

Shortcode introduction

Shortcodes are useful when you want to be able add more complex elements inline into the normal content editor.

A shortcode in it's simplest for would look like this:

```
function my_shortcode( ){
    return "This is a shortcode";
}
add_shortcode( 'my_shortcode', 'my_shortcode' );
```

It would output the text `"This is a shortcode"` and you would use it by writing [my_shortcode] in the content editor.

Button shortcode

Here is an example of a simple button short code:

```
<?php
function my_button_shortcode( $atts ) {

    // Parse the input attributes and assgn default values for the
    // attributes that are not specified on the shortcode
    $a = shortcode_atts( array(
        'id' => '',
        'url' => '#',
        'class' => '',
        'text' => ''
    ), $atts );

    // Open the anchor tag and add role=button for better accessibility
    $btn_html = '<a role="button"';

    // Add the href(link) attribute
    $btn_html .= ' href="' . $a['url'] . '"';

    // Add id attribute to output if specified
    if ( !empty( $a['id'] ) ) {
        $btn_html .= ' id="' . $a['id'] . '"';
    }

    $btn_classes = 'button';


    // Add class attribute to output
    $btn_html .= ' class="button ' . ( !empty(a['class']) ? $a['class'] : '' ) . '"';

    // Close opening anchor tag
```

```
    $btn_html .= '>'.
        // Add button text
        $a['text'].
        // Add closing anchor tag
        '</a>'."\n";

    return $btn_html;
}
add_shortcode( 'button', 'my_button_shortcode' );
```

This shortcode can be used by typing `[button url="/my-other-page" id="my-other-page-button" class="dark" text="Click me!"]` into the editor and will output the following HTML:

```
<a role="button" href="/my-other-page" id="my-other-page-button"
class="button dark">Click me!</a>
```

Chapter 55: Sidebars

Syntax

- register_sidebar($args)
- get_sidebar(string $name = null)

Parameters

Parameter	Details
$args	*(string | array) (Optional)* Builds sidebar based on the `name` and `id` vvalues
$name	*(string) (Optional) The name of the specialised sidebar. Default value: null

Remarks

Argument options are:

- **name** - Sidebar name *(default: localized 'Sidebar' and numeric ID)*.
- **id** - Sidebar id - Must be all in lowercase, with no spaces *(default: a numeric auto-incremented ID)*. If you do not set the id argument value, you will get `E_USER_NOTICE` messages in debug mode, starting with version 4.2.
- **description** - Text description of what/where the sidebar is. Shown on widget management screen. (Since 2.9) *(default: empty)*
- **class** - CSS class to assign to the Sidebar in the Appearance -> Widget admin page. This class will only appear in the source of the WordPress Widget admin page. It will not be included in the front end of your website. **Note**: The value `sidebar` will be prepended to the class value. For example, a class of `tal` will result in a class value of `sidebar-tal`. *(default: empty)*.
- **before_widget** - HTML to place before every widget *(default:* `<li id="%1$s" class="widget %2$s">`*)* **Note**: uses `sprintf` for variable substitution
- **after_widget** - HTML to place after every widget *(default:* `</li>\n`*)*.
- **before_title** - HTML to place before every title *(default:* `<h2 class="widgettitle">`*)*.
- **after_title** - HTML to place after every title *(default:* `</h2>\n`*)*.

Examples

Registering sidebars

In your `functions.php` you can register new sidebars with this code

```
/**
```

```
 * Registers sidebars
 *
 * @param array Array with default or specified array values
 * @since       1.0.0
 */
if ( function_exists( 'register_sidebar' ) ) {
    register_sidebar( array (
        'name'          => esc_html__( 'Primary Sidebar', 'mytheme'),
        'id'            => 'primary-widget-area',
        'description'   => esc_html__( 'The Primary Widget Area', 'mytheme'),
        'before_widget' => '<div id="%1$s" class="widget %2$s">',
        'after_widget'  => '</div>',
        'before_title'  => '<div class="sidebar-widget-heading"><h3>',
        'after_title'   => '</h3></div>',
    ) );

    register_sidebar( array (
        'name'          => esc_html__( 'Secondary Sidebar', 'mytheme'),
        'id'            => 'secondary-widget-area',
        'description'   => esc_html__( 'The Secondary Widget Area', 'mytheme'),
        'before_widget' => '<div id="%1$s" class="widget %2$s">',
        'after_widget'  => '</div>',
        'before_title'  => '<div class="sidebar-widget-heading"><h3>',
        'after_title'   => '</h3></div>',
    ) );
}
```

You can add as many sidebars as you want to.

Get Sidebar

You can also create your own sidebar file in the theme to call it on different templates. Copy and paste sidebar.php of current theme and change the name (i.e. sidebar-book.php)

In the template you can call this sidebar using `get_sidebar('book')`. Using this you can call different sidebars on different pages.

Chapter 56: Site Migration

Syntax

- OLD_SITE - The old site being migrated (eg: http://localhost/example)
- NEW_SITE - The new site to which to migrate (eg: https://example.com

Examples

Updating the tables with a new URL

```
UPDATE wp_options SET option_value = replace(option_value, 'OLD_SITE, 'NEW_SITE') WHERE
option_name = 'home' OR option_name = 'siteurl';
UPDATE wp_posts SET guid = replace(guid, 'OLD_SITE','NEW_SITE');
UPDATE wp_posts SET post_content = replace(post_content, 'OLD_SITE', 'NEW_SITE');
```

Chapter 57: Taxonomies

Syntax

- register_taxonomy($taxonomy, $object_type, $args);

Parameters

Parameter	Details
$taxonomy	*(string) (required)* The name of the taxonomy. Name should only contain lowercase letters and the underscore character, and not be more than 32 characters long (database structure restriction).
$object_type	*(array/string) (required)* Name of the object type for the taxonomy object. Object-types can be built-in Post Type or any Custom Post Type that may be registered.
$args	*(array/string) (optional)* An array of Arguments.

Examples

Example of registering a taxonomy for genres

```
<?php
// hook into the init action and call create_book_taxonomies when it fires
add_action( 'init', 'create_book_taxonomies', 0 );

// create taxonomy genres for the post type "book"
function create_book_taxonomies() {
    // Add new taxonomy, make it hierarchical (like categories)
    $labels = array(
        'name'              => _x( 'Genres', 'taxonomy general name' ),
        'singular_name'     => _x( 'Genre', 'taxonomy singular name' ),
        'search_items'      => __( 'Search Genres' ),
        'all_items'         => __( 'All Genres' ),
        'parent_item'       => __( 'Parent Genre' ),
        'parent_item_colon' => __( 'Parent Genre:' ),
        'edit_item'         => __( 'Edit Genre' ),
        'update_item'       => __( 'Update Genre' ),
        'add_new_item'      => __( 'Add New Genre' ),
        'new_item_name'     => __( 'New Genre Name' ),
        'menu_name'         => __( 'Genre' ),
    );

    $args = array(
        'hierarchical'      => true,
        'labels'            => $labels,
        'show_ui'           => true,
```

```
        'show_admin_column' => true,
        'query_var'         => true,
        'rewrite'           => array( 'slug' => 'genre' ),
    );

    register_taxonomy( 'genre', array( 'book' ), $args );

}
?>
```

You can define custom taxonomies in a themes's `functions.php` template file:

Add category in page

You can also add same custom created taxonomy into post type page using below code.

```
function add_taxonomies_to_pages() {
     register_taxonomy_for_object_type( 'genre', 'page' );
 }
add_action( 'init', 'add_taxonomies_to_pages' );
```

Add above code into your theme's functions.php file. Same way you can add custom or default `post_tag` into post type page.

To get pages using custom taxonomy query need to add below code in same file.

```
if ( ! is_admin() ) {
     add_action( 'pre_get_posts', 'category_and_tag_archives' );
 }

function category_and_tag_archives( $wp_query ) {
    $my_post_array = array('page');
    if ( $wp_query->get( 'category_name' ) || $wp_query->get( 'cat' ) )
    $wp_query->set( 'post_type', $my_post_array );
}
```

Add Categories and Tags to Pages and insert them as class into

```
// add tags and categories to pages

function add_taxonomies_to_pages() {
 register_taxonomy_for_object_type( 'post_tag', 'page' );
 register_taxonomy_for_object_type( 'category', 'page' );
 }
add_action( 'init', 'add_taxonomies_to_pages' );
 if ( ! is_admin() ) {
 add_action( 'pre_get_posts', 'category_and_tag_archives' );

 }
function category_and_tag_archives( $wp_query ) {
$my_post_array = array('post','page');

 if ( $wp_query->get( 'category_name' ) || $wp_query->get( 'cat' ) )
 $wp_query->set( 'post_type', $my_post_array );
```

```
 if ( $wp_query->get( 'tag' ) )
 $wp_query->set( 'post_type', $my_post_array );
}

// add tags and categorys as class to <body>

function add_categories_and_tags( $classes = '' ) {
    if( is_page() ) {
        $categories = get_the_category();
        foreach( $categories as $category ) {
            $classes[] = 'category-'.$category->slug;
        }
        $tags = get_the_tags();
        foreach( $tags as $tag ) {
            $classes[] = 'tag-'.$tag->slug;
        }
    }
return $classes;
}
add_filter( 'body_class', 'add_categories_and_tags' );
```

Add Categories and Tags to Pages and insert as class into

You can add this code to your custom functions.php file:

```
// add tags and categories to pages

function add_taxonomies_to_pages() {
    register_taxonomy_for_object_type( 'post_tag', 'page' );
    register_taxonomy_for_object_type( 'category', 'page' );
}
add_action( 'init', 'add_taxonomies_to_pages' );

if ( ! is_admin() ) {
    add_action( 'pre_get_posts', 'category_and_tag_archives' );
}

function category_and_tag_archives( $wp_query ) {
$my_post_array = array('post','page');

    if ( $wp_query->get( 'category_name' ) || $wp_query->get( 'cat' ) )
    $wp_query->set( 'post_type', $my_post_array );

    if ( $wp_query->get( 'tag' ) )
    $wp_query->set( 'post_type', $my_post_array );

}

// add tags and categorys as class to <body>

function add_categories_and_tags( $classes = '' ) {
    if( is_page() ) {
        $categories = get_the_category();
        foreach( $categories as $category ) {
            $classes[] = 'category-'.$category->slug;
        }
        $tags = get_the_tags();
        foreach( $tags as $tag ) {
            $classes[] = 'tag-'.$tag->slug;
```

```
        }
    }
return $classes;
}
add_filter( 'body_class', 'add_categories_and_tags' );
```

Works perfect, tested in **WordPress 4.8**

Chapter 58: Template hierarchy

Remarks

Plugins for debugging in WordPress:

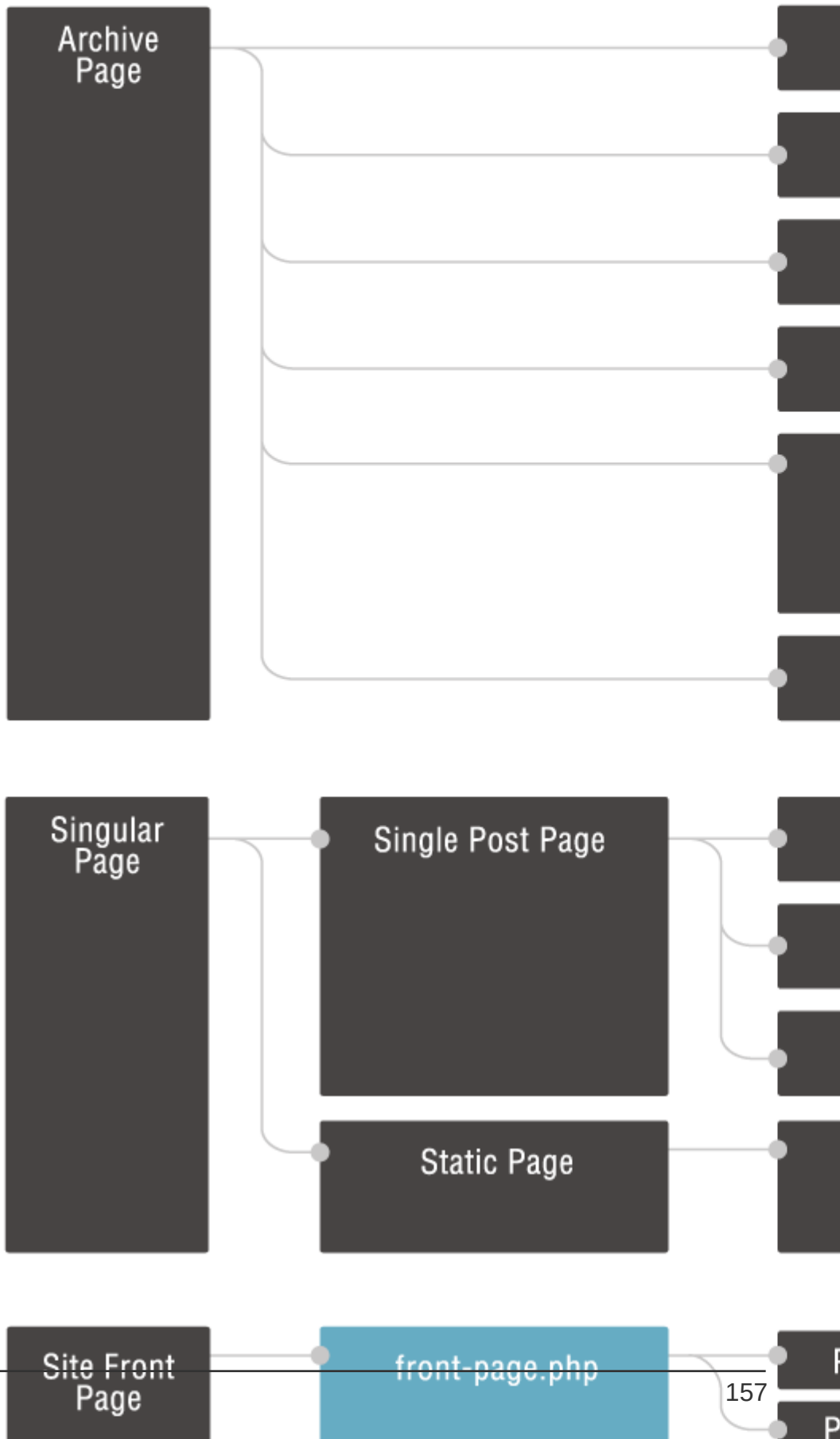
- https://wordpress.org/plugins/query-monitor/
- https://wordpress.org/plugins/debug-bar/
- https://wordpress.org/plugins/debug-bar-console/
- https://wordpress.org/plugins/kint-debugger/
- https://wordpress.org/plugins/rest-api-console/

Examples

Introduction

One of the most important things to learn when you are making a WordPress theme is the WordPress Template hierarchy for themes. The template hierarchy defines what template file that will be loaded for each request and in what order. If the first template does not exist in the hierarchy WordPress will try to load the next one and so on until you end up in `index.php`.

To describe the template hierarchy in detail the best way is of course to use an image with the full structure:


Archive Page
Singular Page
Single Post Page
Static Page
Site Front Page
front-page.php

`category-books.php` would be used only for the category with the slug `book`. Another example is `page-$id.php` that only targets a page with a specific ID, for example `page-41.php` would target only the page with the ID 41.

After the templates which targets specific types or posts we get to the generic type templates, like archive.php for all archive pages or `page.php` for all pages. But remember, those will only be used if the current page does not match any of the templates that are higher in the hierarchy.

Lastly, if WordPress couldn't find any matching templates in the template directory, the last fallback is always `index.php` which is the only required template file in a WordPress theme.

Debugging

Its easy to get lost while debugging the hiearchy. You can use PHP's built in command `debug_backtrace`.

Put the next snippet inside any template you want to debug and view the generated page:

```
<!--
<?php print_r( debug_backtrace() ) ?>
-->
```

Chapter 59: template_include

Parameters

Parameter	Explanation
`$template`	Passes one parameter to the filter, `$template` is the current path to the appropriate file for the post type as found in the active child theme or parent theme (if no child theme in place or child theme has lower ranked templates. See wordpress template hierarchy for more details).

Remarks

You must return `$template` even if not modifying. If this confuses you, look at examples where `apply_filter()` has been used in code

You should not set up variables here for use later, there are better hooks for this.

A useful program flow for this filter is:

1. Check `$template` includes our custom post type name --> template hierarchy!!
2. if not, search our plugin for suitable files --> Its better to point to specific files rather than searching through folders for files. More efficient. But completely up to the developer.
3. return the template.

Examples

Simple example

This filter is very useful. One of the common problems for developers is how to include templates in plugins they develop.

The filter is applied immediately after wordpress locates the appropriate template in the active child/parent theme using the wp hierarchy.

Be careful to define when you want to modify the template path. In the below example, the code checks to see if the current page is the single view of our custom post type `cpt`.

Simple enough example to get started with!

```
add_filter('template_include', 'custom_function');


function custom_function($template){

    //change a single post template...
```

```
    if( is_singular('cpt')  ){
         $template= 'path/to/another/template_file';
    }


    return $template;

}
```

More Adv example

```
add_filter('template_include', 'custom_function');


function custom_function($template){

    /*
    *     This example is a little more advanced.
    *    It will check to see if $template contains our post-type in the path.
    *    If it does, the theme contains a high level template e.g. single-cpt.php
    *    If not we look in the plugin parent folder for the file. e.g. single-cpt.php
    */


    //check to see if the post type is in the filename (high priority file)
    //return template if it is!

    global $post;

    if( strpos($template, 'single-'.$post->post_type.'php' ) !== false && strpos($template,
'archive-'.$post->post_type.'php' ) !== false ){
        return $template;
    }

    $plugin_path = 'var/etc/wp-content/plugins/plugin'; //include own logic here...

    if( is_singular($post->post_type) && file_exists($plugin_path.'/single-'.$post-
>post_type.'.php')   ){
         $template= $plugin_path.'/single-'.$post->post_type.'.php';
    } elseif ( is_archive($post->post_type) && file_exists($plugin_path.'/archive-'.$post-
>post_type.'.php') ) {
         $template= $plugin_path.'/archive-'.$post->post_type.'.php';
    }

    return $template;
}
```

Chapter 60: The $wpdb Object

Remarks

There are two ways to reference the `$wpdb` object. The first is to use the PHP keyword `global` in order to act on the global instance of the object.

```
global $wpdb;
echo $wpdb->prefix;
// Outputs the prefix for the database
```

The second way to use the `$wpdb` object is to reference PHP's `$GLOBALS` super global variable.

```
echo $GLOBALS['wpdb']->prefix;
// This will also output the prefix for the database
```

The second way is discouraged as it may not be considered the best practice.

Examples

Selecting a variable

In the most basic form, it is possible to select a single variable from a table by calling the object's `get_var` method passing in an SQL query.

```
global $wpdb;
$user = $wpdb->get_var( "SELECT ID FROM $wpdb->users WHERE user_email='foo@bar.com'" );
```

It is very important to note that any untrusted values used in queries *must* be escaped in order to protect against attacks. This can be done using the object's `prepare` method.

```
global $wpdb;
$email = $_POST['email'];
$user = $wpdb->get_var(
    $wpdb->prepare( "SELECT ID FROM $wpdb->users WHERE user_email=%s", $email )
);
if( !is_null( $user ){
    echo $user;
} else {
    echo 'User not found';
}
```

Selecting multiple rows

You can use get_results to get multiple rows from database.

```
global $wpdb;
```

```
$userTable =$wpdb->prefix."users";

$selectUser = $wpdb->get_results("SELECT * FROM $userTable");
```

This will show all users list in array.

Chapter 61: The Admin Bar (aka "The Toolbar")

Remarks

The WordPress Admin Toolbar was added in version 3.1 and contains links to common administrative tasks as well as links to the user's profile and other WordPress information. However, many site owners dislike showing the toolbar by default to all logged-in users and/or want to add their own options to it.

Examples

Removing the Admin Toolbar from all except Administrators

Add the following code to `functions.php` to remove it from everyone except the Administrator user level:

```
add_action('after_setup_theme', 'no_admin_bar');

    function no_admin_bar() {
        if (!current_user_can('administrator') && !is_admin()) {
          show_admin_bar(false);
        }
    }
```

Removing the Admin toolbar using filters

Another way to hide admin bar is to add

```
if ( !current_user_can( 'manage_options' ) ) {
    add_filter( 'show_admin_bar', '__return_false' , 1000 );
}
```

The users who don't have privileges to access Settings page, won't be able to see the admin bar.

How to Remove WordPress Logo From Admin Bar

Developers can use **admin_bar_menu** action to remove items from WordPress admin bar or toolbar.

```
add_action('admin_bar_menu', 'remove_wp_logo_from_admin_bar', 999);
function remove_wp_logo_from_admin_bar( $wp_admin_bar ) {
    $wp_admin_bar->remove_node('wp-logo');
}
```

Above code removes WordPress logo from the admin bar. All you need to do is paste the code

inside your functions.php file.

The parameter passed to remove_node method is the ID of the node you wish to remove. ID's can be found in the HTML source code of WordPress page with a Toolbar on it. For example, the li element's ID for the WordPress Logo on the left in the Toolbar is "wp-admin-bar-wp-logo":

```
<li id="wp-admin-bar-wp-logo" class="menupop"> … </li>
```

Remove "wp-admin-bar-" from the li's ID to get the ID of node. From this example the node ID is "wp-logo".

You can use browser inspector tools to find out the node ID's of various items or nodes on your admin bar.

Add your custom logo and custom link on admin login page

You can add below hooks to add your own logo and link to replace default wordpress logo.

To add custom logo

```
function custom_login_logo() {
echo '<style type="text/css">
h1 a { background-image: url('.get_bloginfo('template_directory').'/images/custom-logo.png)
!important; background-size : 100% !important; width: 300px !important; height : 100px
!important;}
</style>';
}
add_action('login_head', 'custom_login_logo');
```

To add custom logo link

```
add_filter( 'login_headerurl', 'custom_loginlogo_url' );
function custom_loginlogo_url($url) {
    return home_url();
}
```

Chapter 62: The Loop (main WordPress loop)

Examples

Basic WordPress loop structure

Each time WordPress loads the page, it will run *main loop*.

The loop is the way to iterate over all elements related to the page you are currently on.

Main loop will work on a global `WP_Query` object. The query has a globalized method `have_posts()`, that allows us to loop through all results. Finally inside the loop you can call `the_post()` method (also as a global function), which sets global post object to the current post inside the loop, and sets the postdata to the current post. Thanks to this you can call functions like `the_title`, `the_content`, `the_author` (*template tags*) directly inside the loop.

For example if you are on posts lists, main loop will contain a query object with all posts.

If you are on single post (or page), it will contain a query with single post (page) you are currently on.

```
if ( have_posts() ) :
    while ( have_posts() ) :
        the_post();
        var_dump( $post );
    endwhile;
endif;
```

Alternative loop syntax

You can also use loop with curly brackets like this:

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

        the_post();
        var_dump( $post );

    }
}
```

Handling no items in the loop

If you want to handle such scenario just add an `if/else` statement.

```
if ( have_posts() ) : while ( have_posts() ) :

    the_post();
    var_dump( $post );
```

```
endwhile; else :

    __('This Query does not have any results');

endif;
```

Chapter 63: the_title()

Introduction

This function returns the title of the current post.

Syntax

- the_title($before, $after, $echo);

Parameters

Parameter	Details
$before	(string) (optional) Text to place before the title.
$after	(string) (optional) Text to place after the title.
$echo	(Boolean) (optional) Display the title or return it for use in PHP

Remarks

- For the parameter $echo, use true to display the title and use false to return it for use in PHP
- Please note that the_title can only be used in loops, if you want to use it outside of loops, please use get_the_title

Examples

Simple use of the_title

Code

```
the_title( );
```

Output

The title of the current post or page

Printing the title with code before and after

Code

```
the_title( '<h1>', '</h1>' );
```

Output

The title of the current post or page in h1 tags

Chapter 64: Themes

Introduction

WordPress themes are the front-end of your website. They are what people see when they visit the site. There are thousands of themes to choose from, paid and free versions. You can even create your own custom theme with just a couple of necessary files.

Examples

WordPress Themes

How To Choose A Theme

Each WordPress install comes with a pre-installed theme. You manage your themes from the Dashboard. Go to Appearance > Themes to install, preview, delete, activate, and update Themes. The current theme is found in the upper left corner of this menu.

Hovering over the theme image reveals a 'Theme Details' button. This button provides information about the theme, such as the version and description. Clicking on the current theme image gives you access to customize specific theme settings, like the title.

Update Available

If updates are available for installed themes, you'll find a message that informs you that a new version is available. You should be able to view the new version details or update now.

- View Version Details

  Clicking the version details link will navigate you to a page from the WordPress Theme Directory. Here you'll find the details for the upgrade version.

- Update Now

  Clicking the update now link will install the Theme upgrade. Themes can also be upgraded from the Administration > Dashboard > Updates screen.

In addition to your current Theme, the Manage Themes screen also shows the other Themes that are installed but currently inactive. Each Theme is represented by a small screenshot. Hovering over these images reveals 'Theme Details', 'Activate', and 'Live Preview' buttons. You'll also be able to upgrade or delete inactive themes from this page. Each page on this screen will display up to 15 Theme screenshots at a time.

- Activate

Clicking this link makes this the Current Theme.

- Live Preview

  Clicking this link displays a preview of blog with this specific theme version.

- Delete

  Clicking this link completely deletes this Theme, include all Theme files and folders. Anything not backed up will be lost forever.

- Update Available

  Refer to the Update Available section above.

Install Themes

Listed below are several ways to install Themes:

- Automated Theme Installer

  This can be used to install Themes from the WordPress Theme Directory. Go to Administration > Appearance > Themes to find the Appearance Themes Screen. Click the Add New button. From here you'll find Themes to use that are free of change. At the top of this screen there is search feature with three available methods to find a new Theme; Filter, Keyword, and Attribute search.

- Using The Upload Method

  The upload method installs a Theme via a ZIP file. All the Themes in the WordPress Theme Directory can be installed this way. After downloading the ZIP file, visit Administration > Appearance > Themes, and click the Add New button. Next, click the Upload Theme link. Browse for the ZIP file and click Install Now. To finish making this the Current Theme click the Activate link.

- Using The FTP Method

  To install a Theme with the FTP Method you must first download the Theme files to your local computer. Extract the contents of ZIP file, preserving the file structure, and add them to a new folder. If there are instructions from the Theme author be sure to follow them.

  Use an FTP client to access your site's web server. Add the uploaded Theme files to your wp-content/themes directory provided by WordPress. If need be, create a folder to contain your new Theme inside the wp-content/themes directory. An example of this would be, if your Theme is named Test it should live in wp-content/themes/test.

  Go to Administration > Appearance > Themes, and click the Activate link select the Theme as your Current Theme.

- Installing With cPanel

cPanel control panels offer another method to install Themes with a ZIP or GZ files. In the cPanel Manager, If WordPress is installed, go to your Themes folder. The path would look similar to 'public_html/wp-content/themes'. Click on Upload file(s) and upload the ZIP file. Select the ZIP file in cPanel and click on Extract File Contents in the panel to the right to uncompress that file.

Go to Administration > Appearance > Themes, and click the Activate link select the Theme as your Current Theme.

All information listed above is according to the WordPress Codex. It has been shortened for brevity. The original source material can be found here. Or for more information visit codex.wordpress.org.

Creating A Custom Theme

These instructions create a very basic, minimum standards compliant WordPress Theme.

The first step is to create a new theme folder inside your WordPress themes directory. The correct path will be: > wp-content > themes > To create a valid theme, WordPress themes require at least these two files live there:

- index.php
- style.css

Your stylesheet should contain a comment that alerts WordPress that a theme exists here.

```
/*
Theme Name: <theme name>
Author: <author name>
Description: <description goes here>
Version: <theme version #>
Tags: <tag to id theme>
*/
```

Your theme has now been created. Go to the WordPress dashboard, and click on Appearance > Themes, to activate it.

Chapter 65: Update WordPress Manually

Examples

VIA FTP

1. Download the desired version of WordPress from www.wordpress.org to your local computer and unzip the file.
    - Also keep a backup of your current version... just in case.
2. Connect to your website with your favorite FTP client (FileZilla is popular and easy, but any FTP client will be fine).
    - *Instructions for this are outside the scope of WordPress, but may be found at a future date in the proposed FTP topic*.
3. Upload the folders (and their contents) titled "wp-admin" and "wp-includes" to their matching directories on your server. Be sure to overwrite the current folders.
    - You can omit uploading the "wp-content" folder unless you choose to use one of the themes included. If you wish to update/upload the default themes included with your chosen version you should upload this folder as well.
4. Upload the individual files in the home folder (index.php, wp-*.php, etc).
    - You can omit the files titled "liscense.txt" and "readme.html" as they are not required to function and they can be used as methods of determining your WP version for security exploits.
5. Visit and log into your website in order to perform any required database updates.
    - Not all WP updates have DB changes, but some do.

Note: This method will create orphaned files that can/will build up over time and may present security risks. Be sure to do a file comparison after completion and delete old files off your server from previous WP versions which are no longer in use.

Chapter 66: WordPress Plugin creation

Introduction

WordPress plugins should have a focus on the server logic and/or admin parts of your website application. Good plugins are like good apps, they do one thing really well. They are intended to enhance and automate parts of the CMS in a modular way, since you can activate and deactivate them. Good plugins make use of WordPress core actions, filters, and existing javascript and css frameworks.

Examples

Minimal Setup of a Plugin Folder and Files

First step of creating a plugin is creating the folder and file which the plugin will load from.

Plugins are located in `/wp-content/plugins/`.

The WordPress standard is to create a folder and filename that mirror each other like so:

```
/wp-content/plugins/myplugin/
/wp-content/plugins/myplugin/myplugin.php
```

After creating your plugin file you need to start your plugin with a `Plugin Header`. This allows WordPress to scan your plugin file and store the meta data about the plugin, and allow users to use this and determine if they want your plugin active, or inactive. Copy this template into the top of your main plugin file you created, and modify it as needed:

```
<?php
/**
 * Plugin Name: PLUGIN-NAME
 * Plugin URI: HTTP-LINK-TO-WEBSITE-PLUGIN-PAGE-OR-REPO
 * Description: BREIF DESCRIPTION - KEEP IT SHORT
 * Author: WORDPRESS-DOT-ORG-USERNAME
 * Version: 0.0.1
 * Author URI: HTTP-LINK-TO-MAINTAINER
 * License: GNU General Public License v2 or later
 * License URI: http://www.gnu.org/licenses/gpl-2.0.html
 * Text Domain: short_prefix
 */

// Begin custom PHP WordPress plugin work
```

Note that WordPress plugins should typically be licensed as GPL. Licensing should not be discussed as part of this topic though.

At this point, you should already by able to see your new plugin in the WordPress Admin area. In a standard setup you would locate this area at `/wp-admin/plugins.php`. Go ahead and activate your

plugin, and you are ready move into the next steps of building your plugin!

Just to end our example on something actionable, you could now add the following to the bottom of your plugin file:

```
die('My custom plugin is loaded : '. __FILE__);
```

Refreshing your site after this change should result in all pages printing this text. *Never do this in production (live) sites, and always remember to take this back out before continuing.*

Chapter 67: Wordpress theme and child-theme development

Introduction

Wordpress is a widely used CMS for creating simple information websites but also for creating more sophisticated websites and even small webshops.

Wordpress makes use of themes. These themes are used for creating the lay-out and content functionality of a Wordpress website. The themes can be found all over the internet.

Each thme has his own unique functionality and lay-out but sometimes it's hard to find just the right theme for a website. Luckily we're also able to create our own theme.

Examples

Developing your own theme

A wordpress theme consists two types of files. The basic files that each theme has and the files that define the theme's layout and functionality. This second group i'm going to call the theme specific files.

The basic theme files
The basic theme files are the files that are used to setup and register a theme. In the list below i will shortly describe each file and its usage. Later on i'll add the most basic example files that are needed to set up your own wordpress theme.

- `functions.php`: The functions.php file is used to register all the functions, sidebars, scripts and includes of the theme. In this file you're able to, for example, include CSS files, JS files, etc.
- `Header and footer`: The header and footer files (header.php and footer.php) are the files that are used to call on the header and the footer. The header and footer file for example hold the link to the wordpress back-end system.
- `index.php`: The index.php file is the file that creates the default-page template. In this file you can see, edit and remove pieces of this default-template lay-out.
- `single.php`: The single.php file is the file that creates the single posts template page. Just like the default-template for the pages but now for the single post pages.
- `format.php` The format.php file is the file that builds the content-text template from a page. So if you would have a home page and you would edit it from the back-end by adding a text. This file creates the standard markup of this text.
- `404.php` The 404.php file creates the 404 template. This file consists of the basic lay-out of this page.
- `archive.php` The archive.php file creates the lay-out of the archive page.
- `style.css` The basic stylesheet file.

So in this list you can see all the **required** files for the set up of your very own Wordpress theme. Now lets take a look at some files that you're able to create if you want to but are **not required** files for a wordpress theme. These files are mostly template files and other functional extentions.

Custom page templates

`page-<your own name>.php`: In a Wordpress theme you're able to create multiple page templates. by creating new page template files. A standard page template file consists of the following name attributes. `page name of the template` and `.php` If for example you would like to create a new page template for your blog page you could call it `page-blog.php` Wordpress automaticly reads the file and adds the file to the choose template menu. Do make sure that you've atleast included the `get_header()` and `get_footer()` functions. Also make sure you name your template in a comment at the top of the file by adding the following example.

```
<?php
    /*
     * Template Name: Homepage Template
     */
    get_header();
?>
```

Custom single post page templates

`single-<your own name>.php`: In a Wordpress theme just like the page template described above you're also able to create your own single posts page templates. Just like the page template the file consists of three parts `single` for declaring it's a single post page `<your name of the template>` and the file extention `.php`. Just like the page template minimum requirements to make sure Wordpress reads the new template are adding the functions `get_header()` and `get_footer()`. And ofcourse also adding your template name like the example below

```
<?php

/*
* Template Name: Post Portfolio
* Template Post Type: post, page
*/

?>
```

We also indicate the `Template post type:` wich stands for the kind of template it is, in this case post and page.

Custom post text templates

`format -<your own name>.php`: In a Wordpress theme you're also able to create post output templates. These format templates are the lay-out and contents of a post. For example if in some cases you want the post to only show the content or the title of the post you can use these templates to create those kind of adjustments. Since these kind of templates are only formatting the post back-ends content that was created by a user we don't need to include `get_header()` and `get_footer()` since these are already defined in the pages templates. Do make sure your template is able to recognize a post by using the following basic example.

```
<div>
```

```
    <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
    </article>
</div>
```

So now that we know something about the basic files and some of the many template specific files it's time to start talking about sidebars and widgets. In the future this will be added together with a start on the step to step tutorial on creating a very own Wordpress theme.

Chapter 68: wp_get_current_user()

Syntax

- wp_get_current_user()

Examples

Getting the current user

Getting all the information of current user in wordpress using the predefined function `wp_get_current_user();`

```
<?php

        $current_user = wp_get_current_user();

        echo "Username :".$current_user->user_login;
        echo "Username :".$current_user->ID;
        echo "Username :".$current_user->user_pass;
        echo "Username :".$current_user->user_nicename;
        echo "Username :".$current_user->user_email;
        echo "Username :".$current_user->user_url;
        echo "Username :".$current_user->user_registered;
        echo "Username :".$current_user->user_activation_key;
        echo "Username :".$current_user->user_status;
        echo "Username :".$current_user->display_name;

    ?>
```

Use foreach loop to get user info from wp_get_current_user()

```
$user = wp_get_current_user();

foreach($user->data as $key=>$user_data){
    if($key == 'user_pass' || $key == 'user_activation_key' || $key=='user_status'){}
    else{
        $nice_key = ucfirst(str_replace('_', ' ', $key));

        if($key == 'user_registered'){
            $user_data = date_i18n(get_option('date_format'), strtotime($user_data));
        }

    echo $nice_key . ' : ' . $user_data . '<br />';
   }
}
```

Chapter 69: wp_get_current_user()

Examples

Get current loggedin user information

Retrieves the information pertaining to the currently logged in user, and places it in the global variable $current_user

This function does not accept any parameters.

Usage:

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

Example:

```
  <?php
  $current_user = wp_get_current_user();

  echo 'Username: ' . $current_user->user_login . "\n";
  echo 'User email: ' . $current_user->user_email . "\n";
  echo 'User level: ' . $current_user->user_level . "\n";
  echo 'User first name: ' . $current_user->user_firstname . "\n";
  echo 'User last name: ' . $current_user->user_lastname . "\n";
  echo 'User display name: ' . $current_user->display_name . "\n";
  echo 'User ID: ' . $current_user->ID . "\n";
  ?>
```

To determine if a visitor is logged in or not, you can use is_user_logged_in() before, then get the current user info if the visitor it's logged in:

```
<?php
if ( is_user_logged_in() ) {
    $current_user = wp_get_current_user();

    echo 'Welcome, ' . $current_user->user_login . '!';
} else {
    echo 'Welcome, visitor!';
}
?>
```

Chapter 70: WP_Query() Loop

Introduction

WP_Query to query for posts, pages and custom post types. You will get list for specific posts and pages or custom post types. WP_Query allows you to pull posts from the database according to your criteria.

Examples

Retrieve latest 10 posts

```
$args = array(
    'post_type'=>'post',
    'posts_per_page' =>'10'
);
$latest_posts_query = new WP_Query( $args );
if($latest_posts_query->have_posts()) :
while ( $latest_posts_query-> have_posts()) : $latest_posts_query->the_post();
//Get post details here
endwhile;
endif;
```

Chapter 71: WP-CLI

Introduction

WP-CLI is a set of command-line tools for managing WordPress installations. You can update plugins, configure multisite installs and much more, without using a web browser.

Examples

Manage themes

Get a list of themes.

```
$ wp theme list
```

Install the latest version from wordpress.org and activate

```
$ wp theme install twentysixteen --activate
```

Install from a local zip file

```
$ wp theme install ../my-theme.zip
```

Install from a remote zip file

```
$ wp theme install http://s3.amazonaws.com/bucketname/my-
theme.zip?AWSAccessKeyId=123&amp;Expires=456&amp;Signature=abcdef
```

Get details of an installed theme

```
$ wp theme get twentysixteen --fields=name,title,version
```

Get status of theme

```
$ wp theme status twentysixteen
```

Manage plugins

Get a list of plugins

```
$ wp plugin list
```

List active plugins on the site.

```
$ wp plugin list --status=active --format=json
```

List plugins on each site in a network.

```
$ wp site list --field=url | xargs -I % wp plugin list --url=%
```

Activate plugin

```
$ wp plugin activate hello-dolly
```

Deactivate plugin

```
$ wp plugin deactivate hello-dolly
```

Delete plugin

```
$ wp plugin delete hello-dolly
```

Install the latest version from wordpress.org and activate

```
$ wp plugin install bbpress --activate
```

Manage WP-CLI itself

Display the version currently installed.

```
$ wp cli version
```

Check for updates to WP-CLI.

```
$ wp cli check-update
```

Update WP-CLI to the latest stable release.

```
$ wp cli update
```

List all available aliases.

```
$ wp cli alias
```

Print various details about the WP-CLI environment.

```
$ wp cli info
```

Dump the list of installed commands, as JSON.

```
$ wp cli cmd-dump
```

Download, install, update and manage a WordPress install.

Download WordPress core

```
$ wp core download --locale=nl_NL
```

Install WordPress

```
$ wp core install --url=example.com --title=Example --admin_user=supervisor --
admin_password=strongpassword --admin_email=info@example.com
```

Display the WordPress version

```
$ wp core version
```

Transform a single-site install into a WordPress multisite install.

```
$ wp core multisite-convert
```

Install WordPress multisite from scratch.

```
$ wp core multisite-install
```

Manage users

List user IDs

```
$ wp user list --field=ID
```

Create a new user.

```
$ wp user create bob bob@example.com --role=author
```

Update an existing user.

```
$ wp user update 123 --display_name=Mary --user_pass=marypass
```

Delete user 123 and reassign posts to user 567

```
$ wp user delete 123 --reassign=567
```

Perform basic database operations using credentials stored in wp-config.php

Create a new database.

```
$ wp db create
```

Drop an existing database.

```
$ wp db drop --yes
```

Reset the current database.

```
$ wp db reset --yes
```

Execute a SQL query stored in a file.

```
$ wp db query < debug.sql
```

Chapter 72: WP-Cron

Examples

wp_schedule_event() example

```
// register activation hook
register_activation_hook( __FILE__, 'example_activation' );

// function for activation hook
function example_activation() {
    // check if scheduled hook exists
    if ( !wp_next_scheduled( 'my_event' )) {
         // Schedules a hook
         // time() - the first time of an event to run ( UNIX timestamp format )
         // 'hourly' - recurrence ('hourly', 'twicedaily', 'daily' )
         // 'my_event' - the name of an action hook to execute.
         wp_schedule_event( time(), 'hourly', 'my_event' );
    }
}

add_action( 'my_event', 'do_this_hourly' );

// the code of your hourly event
function do_this_hourly() {
   // put your code here
}

// register deactivation hook
register_deactivation_hook(__FILE__, 'example_deactivation');

// function for deactivation hook
function example_deactivation() {
    // clear scheduled hook
    wp_clear_scheduled_hook( 'my_event' );
}
```

Important: the WordPress cron runs only when some page of your website is hit. So, for website with low trafic you need to setup the cron on your hosting to hit pages.

custom recurrence interval in wp_schedule_event()

```
// this function add custom interval (5 minutes) to the $schedules
function five_minutes_interval( $schedules ) {
      $schedules['five_minutes'] = array(
            'interval'  => 60 * 5,
            'display'   =>  '5 minutes';
    );
    return $schedules;
}

// add a custom interval filter
add_filter( 'cron_schedules', 'five_minutes_interval' );
```

```
// Schedules a hook
wp_schedule_event( time(), 'five_minutes', 'my_event' );
```

Credits

S. No	Chapters	Contributors
1	Getting started with WordPress	4444, A. Raza, Andrew, animuson, Anupam, Chris Fletcher, Ciprian, Community, Florida, James Jones, JonasCz, Leo F, Marc St Raymond, Mayank Gupta, Milap, nus, Panda, rap-2-h, Seth C., Shubham, Trevor Clarke, vajrasar
2	Actions and Filters	David, Ihor Vorotnov, Mrinal Haque, Trying Tobemyself
3	Add Shortcode	purvik7373, RamenChef
4	Add/remove contact info for users with user_contactmethods filter hook	Petar Popovic, RamenChef
5	add_action()	Abel Melquiades Callejo, Waqas Bukhary
6	add_editor_style()	Gabriel Chi Hong Lee
7	add_menu_page()	brasofilo, Gabriel Chi Hong Lee
8	add_submenu_page()	Gabriel Chi Hong Lee, theoretisch
9	add_theme_support()	Gabriel Chi Hong Lee
10	Admin Dashboard Widgets	theoretisch
11	AJAX	Andy, Digvijayad, Gaurav Srivastava, GreatBlakes, Nisarg Patel, Ruslan Murarov, stweb
12	Alternating main loop (pre_get_posts filter)	Dawid Urbanski, Petar Popovic
13	Child Theme Basics	Andrei, Razvan Onofrei, Vlad Olaru
14	Create a Post Programmatically	RamenChef, Roel Magdaleno, RRikesh
15	Create Template for Custom Post Type	Ashok G, Egnaro, Joe Dooley, mnoronha
16	Creating a custom template	Petar Popovic

17	Custom exerpts with excerpt_length and excerpt_more	inkista, Petar Popovic, RamenChef
18	Custom Post Types	Caio Felipe Pereira, Dan Devine, J.D., janw, jgraup, Kushal Shah, Omar Khaiyam, Ranuka, theoretisch
19	Customizer Basics (Add Panel, Section, Setting, Control)	4444, Ahmad Awais, RamenChef
20	Customizer Hello World	Dan Green-Leipciger
21	Debugging	barbocc, dingo_d, jgraup
22	Enqueuing scripts	dingo_d, Harshal Limaye, J.D., mbacon40, montrealist, Pelmered, Petar Popovic
23	Enqueuing Styles	dingo_d, Harshal Limaye, Laxmana, mnoronha, montrealist, Petar Popovic, RamenChef, Ruslan Murarov, virtualLast
24	Function : wp_trim_words()	Harshal Limaye
25	Function: add_action()	bosco, RamenChef
26	get_bloginfo()	Abel Melquiades Callejo, Harshal Limaye, HeyCameron, KenB , Nate Beers, RamenChef, Tom J Nowell, virtualLast, Wes Moberly
27	get_home_path()	Ihor Vorotnov
28	get_option()	Gabriel Chi Hong Lee
29	get_permalink()	Gabriel Chi Hong Lee
30	get_template_part()	Dan Devine, Kushal Shah
31	get_the_category()	Gabriel Chi Hong Lee
32	get_the_title()	Gabriel Chi Hong Lee
33	home_url()	dingo_d, Kushal Shah, matthew, Mr. Developer
34	How Can I integrate Markdown editor with Advance Custom Field's repeater Add-on.	Fatbit

35	init	Abel Melquiades Callejo, barbocc
36	Installation and Configuration	Kenyon, Marco Romano, Ping.Chen, S.L. Barth, stig-js, theoretisch, Yuan Lung Luo
37	Making network requests with HTTP API	Jordan, mjangda, Rarst
38	Meta Box	Austin Winstanley
39	Options API	Harshal Limaye, Pat J, RamenChef
40	Plugin development	Angle.R, Ping.Chen
41	Post Formats	Shashank Agarwal
42	Querying posts	dingo_d
43	Remove Auto Line Breaks From Content and Excerpt	Austin Winstanley
44	Remove Version from Wordpress and Stylesheets	jay.jivani, mnoronha, theoretisch
45	REST API	Picard
46	Run WordPress local with XAMPP	Pierre.Vriens, theoretisch
47	Secure your installation	James Jones
48	Security in WordPress - Escaping	Laxmana, the4kman
49	Security in WordPress - Sanitization	Laxmana
50	Shortcode	Ad Wicks, Adam Genshaft, brasofilo, John Slegers, Kylar, Shashank Agarwal
51	Shortcode with attribute	Digvijayad, Firefog, RamenChef
52	Shortcodes	Pelmered
53	Sidebars	dingo_d, Kushal Shah, theoretisch

54	Site Migration	Austin Winstanley
55	Taxonomies	adifatz, Kushal Shah, purvik7373
56	Template hierarchy	jgraup, MarZab, Pelmered, theoretisch
57	template_include	Abel Melquiades Callejo, David, RamenChef
58	The $wpdb Object	Kushal Shah, mcon, stweb
59	The Admin Bar (aka "The Toolbar")	dingo_d, Harshal Limaye, JCL1178, Kushal Shah
60	The Loop (main WordPress loop)	anik4e, Dawid Urbanski
61	the_title()	Gabriel Chi Hong Lee
62	Themes	Jef
63	Update WordPress Manually	KnightHawk
64	WordPress Plugin creation	Seth C.
65	Wordpress theme and child-theme development	Deathstorm
66	wp_get_current_user()	Abel Melquiades Callejo, Benoti, Muhammad Farrukh Faizy
67	WP_Query() Loop	vrajesh
68	WP-CLI	jgraup
69	WP-Cron	stweb