Государственное бюджетное профессиональное образовательное учреждение «Дзержинский педагогический колледж»

**Методические разработки самостоятельных занятий**

**по МДК. 08.01   
ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНТЕРФЕЙСОВ ПОЛЬЗОВАТЕЛЯ**

**Дзержинск – 2017-2021**

|  |  |
| --- | --- |
| **Одобрено на заседании ПЦК преподавателей спец. Информатика**  **Протокол №\_\_\_\_\_\_\_от\_\_\_\_\_\_\_\_**  **Председатель ПЦК \_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_/**  **Составитель: Пучкина Т.С., Руденко Н.А.** | **Методические разработки самостоятельных занятий составлены в соответствии с требованиями федеральных государственных образовательных стандартов среднего профессионального образования по спец. 09.02.07. Информационные системы и программирование** |

Методические разработки самостоятельной работы занятий по МДК. 08.01 Проектирование и разработка интерфейсов пользователя 09.02.07. Информационные системы в программировании. Представленные, в данных методических указаниях, самостоятельных задания направлены на формирование знаний и умений по дисциплине, а так же общих и профессиональных компетенций.

**Установка и настройка WordPress**

# Цель: установить и настроить CMS WordPress на локальный сервер Denwer.

**Основные теоретические сведения**

Что такое CMS?

CMS, от английского Content Management System (система управления контентом), - это программное обеспечение, позволяющее пользователям размещать или изменять уже размещенную на сайте информацию без привлечения разработчиков сайта. Это значит, что пользователю не обязательно обладать навыками программирования или знаниями языка HTML, чтобы, например, опубликовать на своем сайте новость, статью или добавить изображение. Часто наряду с термином CMS можно услышать также термин "движок сайта", которым обычно пользуются веб-мастера в своем профессиональном сленге.

Состоят CMS обычно из двух частей:

* back-office - это часть системы, отвечающая за функциональность и хранение информации;
* front-office - это часть системы, обеспечивающая интерфейс с пользователем.

Как работают CMS?

Принцип работы всех CMS основан на разделении контента (содержания) и дизайна (оформления) сайта. Обычно дизайн сайта меняется редко, тогда как изменения контента могут происходить не только каждый день, но и даже каждый час. Поэтому в своей работе CMS используют так называемые шаблоны - специальные "пустые" заготовки страниц, в которых дизайн сайта уже прописан и осталось лишь наполнить их информацией. Пользователю достаточно воспользоваться специальным WYSIWYG-редактором. Этот редактор по внешнему виду очень похож на привычные текстовые редакторы офисных приложений, поэтому пользователю не составляет особого труда освоить его. А наличие в системе большого количества готовых шаблонов дает возможность выбрать подходящий дизайн буквально в считанные минуты.

Информация хранится в базе данных, например, в MySQL и вызывается из нее при загрузке страниц сайта.

Работа CMS не требует установки дополнительного ПО, поскольку сама система находится на сервере, а доступ к ней осуществляется через обычный интернет-браузер.

Что такое Denwer?

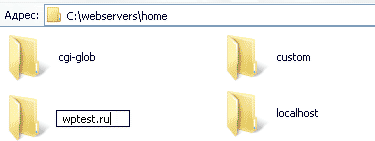
Denwer, по-другому его называют Джентельменский набор WEB разработчика. Денвер это программа, предназначенная для имитации WEB сервера на домашнем компьютере. Используется она для тестового запуска и отладки WEB сайтов. На Denver можно запустить практически любой сайт.

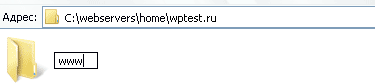
В стандартную комплектацию этого так называемого программного комплекса входят: WEB сервер Apache, интерпретатор языка программирования PHP, интерпретатор языка PERL, база данных MySQL, имитация сервера Email почты, по умолчанию встроен движок phpMyAdmin для управления базами данных MySQL. Это всё что нужно чтобы запустить практически любую современную CMS систему.

**Практическая часть**

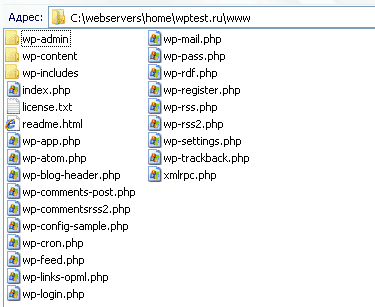
Подготовка

1. Скачиваем Денвер с са[йта http://www.denwer.ru/](http://www.denwer.ru/)
2. Устанавливаем Денвер (подробней про установку Вы можете прочитать тут - [http://www.denwer.ru/base.html).](http://www.denwer.ru/base.html))
3. Далее скачиваем последнюю версию WordPress с сайта - <http://ru.wordpress.org/> I. Установка необходимого ПО
4. Запускаем Денвер для этого открываем папку C:\WebServers\etc, и запускаем Run.exe.
5. Далее в папке C:\WebServers\home\ создаем папку с названием Вашего сайта - например wptest.ru



переходим в эту папку и создаем в ней папку [www.](http://www/) В итоге получаем такой путь: C:\WebServers\home\wptest.ru\www

1. Распакуйте архив с последней версией WordPress в папку C:\WebServers\home\wptest.ru\www



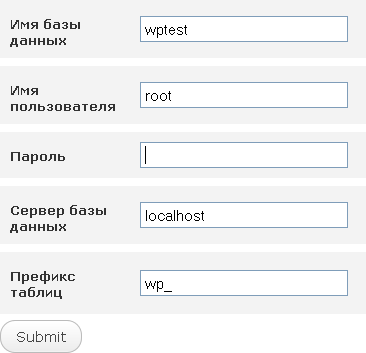
1. Создаем базу данных, для этого в адресной строке Вашего браузера набираем

<http://localhost/Tools/phpMyAdmin/>вводим логин и пароль (по умолчанию имя пользователя - root, пароля нет). В открывшимся окне в поле "Создать новую БД" пишем название базы

данных (например wptest), в соседнем поле выбираем сравнение с utf8\_general\_ci и нажимаем кнопку "создать".



1. Далее запускаем и устанавливаем WordPress - для этого в адресной строке Вашего браузера набираем [http://wptest.ru](http://wptest.ru/) - в появившимся окне нажимаем кнопку "создать файл настроек",

далее кнопку "Вперед!" в появившимся окне заполняем все поля.

В поле "Имя базы данных" вводим название базы данных, в нашем случаи wptest.

В поле "Имя пользователя" вводим имя пользователя базы данных, в нашем случаи root. В поле "Пароль" вводим пароль баззы данных, в нашем случаи оставляем его ПУСТЫМ. В поле "Сервер базы данных" оставляем localhost.

В поле "Префикс таблиц" оставляем wp\_ далее нажимаем кнопку Sumbit, затем в появившимся окне нажимаем кнопку "Запустить установку". Далее вводим заголовок Вашего сайта и Ваш емэйл (на него Вам придет пароль для доступа к админке, а также будет приходить информация о новых комментариях на сайте) и нажимаем кнопку "Установить WordPress". На этом установка WordPress завершена. Переписываем или копируем логин и пароль и входим на

Ваш новый сайт - [http://wptest.ru](http://wptest.ru/) для перехода в админку <http://wptest.ru/wp-admin/> На этом установка завершена.

1. Ознакомление с интерфейсом WordPress

Ознакомьтесь с интерфейсом системы управления контентом WordPress (WP). Посмотрите какие элементы есть в **Консоли** WP, для чего они служат. Также ознакомьтесь с **Настройками** WP.

В настройках найдите опцию **Видимость для поисковых систем** и поставьте «галочку» **Попросить поисковые системы не индексировать сайт** (Данная опция не даёт гарантий, что ПС не будут индексировать ваш ресурс, но на этапе создания сайта её можно включить).

Поставьте в настройках программы необходимый часовой пояс (UTC +9) и выберите подходящий для вас формат даты.

1. Установка темы для вашего сайта (блога)

В пункте меню **Внешний вид** выберите **Темы** и загрузите предложенный преподавателем шаблон (тему). Активируйте тему и ознакомьтесь с её настройками (для разных тем могут быть разные настройки). В настройках темы заполните **Свойства сайта** (**Название сайта** и **Краткое описание**). Добавлением виджетов и меню мы займёмся позже (Прежде чем добавить меню, необходимо создать **Страницы** для вашего сайта).

В интернете можно найти множество бесплатных и платных **тем** для WP, так данная система управления контентом является очень популярной, так же вы можете создавать свои темы. Один из качественных ресурсов, где вы можете скачать темы для WP: wp-templates.ru

1. Добавление страниц и создание меню

Следующий шаг в создании вашего сайта – добавление **Страниц** и в дальнейшем создание **Меню**. Вы должны определиться какие страницы у вас будут основными, т.е. будут главными элементами меню сайта. Для создания страниц вам нужно определиться с предметной областью (тематикой) вашего сайта. В практической работе лучше выбрать тему личного сайта или сайта портфолио, также можете сделать сайт вашей группы.

Создайте 3-4 страницы для ваше сайта, причём одну страницу сделайте дочерней по отношению к другой (для того чтобы был элемент выпадающего меню). Например, страница Портфолио – для неё дочерняя страница Мои достижения или Обо мне.

В зависимости от выбранной тематики сайта наполните страницы информацией. Добавьте тест и картинки. Для добавления картинок их сначала нужно загрузить в **Медиафайлы.** Для каждой картинки добавляйте **Описание** и **Атрибут alt** (это является частью поисковой оптимизации вашего ресурса). При добавление информации на страницы обратите внимание что во встроенном редакторе есть 2 режима: **Визуально** и **Текст**. Режим **Текст** показывает теги и даёт возможность вручную их добавлять и менять.

После добавления страниц создайте меню для вашего сайта. Для этого в **Консоли** выберите **Внешний вид** – **Меню**. В Структуре меню вы можете вручную перетаскивать элементы меню в нужное место и создавать дочерние элементы (многоуровневое меню). После создания структуры меню в **Настройках меню** поставьте «галочку» **Главное меню**.

Когда вы создадите меню обязательно проверьте добавилось ли оно и как работает.

# Контрольные вопросы

* 1. Что такое CMS, как они работают?
  2. Что такое Denwer и для чего используется?
  3. Как установить WordPress на локальный сервер Denwer?
  4. Какие вы знаете основные элементы Консоли WP?
  5. Чем CMS отличаются от конструкторов сайтов?

**Разработка собственной темы**

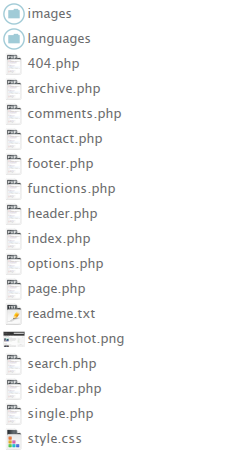
**Цель:** Научится создавать тему для своего сайта

## **Теоретические основы.**

Чтобы тема корректно отображалась в административной панели, нам потребуется завести несколько файлов:

* index.php
* single.php
* page.php
* header.php
* footer.php
* style.css
* screenshot.png
* functions.php

На самом деле было бы достаточно одного файла стилей и индексной страницы, но давайте сразу идти по правильному пути и разделять тему на составные части.



Коротко о каждом: index.php представляет собой главную страницу сайта. В ней всегда содержится цикл вывода последних записей. Конечно, главная страница вашего ресурса может быть какой угодно (тогда создаётся отдельный шаблон), но шаблон index.php также будет использоваться при выводе всех категорий, всех статей автора и подобных вещей. Это лучше держать в уме.

Страница отдельной записи single.php призвана отображать одну публикацию. От page отличается тем, что вторая нужна для вывода статических страниц: страницы контактов, например.

header и footer обычно включаются в каждый шаблон каждой страницы, поскольку их содержимое не меняется от одной странице к другой (за редкими исключениями).

Также обязательно включать в конец первого вывод <?php wp\_head(); ?>, а второго — <?php wp\_footer(); ?> во избежание проблем различной степени тяжести (например, если не выводить wp\_footer, может не отображаться панель администратора на самом блоге).

Стили располагаются в style.css, screenshot.png служит миниатюрой темы при её выборе в админке, а functions.php содержит всю магию: там прописываются пути к стилям, шрифтам и скриптам темы, настройки для виджетов, меню и многих других полезных вещей.

**Практическое задание:**

Возьмем простенький HTML код главной страницы с разметкой шапки сайта, одного анонса поста, правой колонки и футера:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Мой блог</title>

<link href="style.css" rel="stylesheet" type="text/css" />

</head>

<body>

<div id="header">

<div id="logo">

<a href="index.html"><img src="images/shapka.png" alt="" title=""/></a>

</div>

<ul id="menu">

<li><a href="#">Главная</a></li>

<li><a href="#">Контакты</a></li>

<li><a href="#">Рубрика 1</a></li>

<li><a href="#">Рубрика 2</a></li>

</ul>

</div>

<div id="content-main">

<div id="content">

<div class="post-content">

<div class="post-img"><a href="#"><img width="275" height="230" src="images/mini.jpg" alt="Миниатюра поста"/></a></div>

<h2 class="note-title"><a href="#">Заголовок поста</a></h2>

<div class="post-text"> Текст анонса поста</div>

<div class="readmore"><a href="#">Кнопка читать далее</a></div>

</div>

</div>

<div id="saidbar">

<div class="widget-title">Популярные рубрики</div>

<div class="widget">

<ul>

<li><a href="#">Мои услуги</a></li>

<li><a href="#">WordPress</a></li>

<li><a href="#">Сайтостроение</a></li>

<li><a href="#">О заработке</a></li>

<li><a href="#">О SEO</a></li>

<li><a href="#">О YouTube</a></li>

</ul>

</div>

</div>

</div>

<div id="footer">

<p>Все права защищены</p>

</div>

</body>

</html>

Для создания темы оформления необходимо знать функции или как их еще называют — теги WordPress. Мы разберем основные функции темы, со всеми тегами вы можете ознакомиться на официальном сайте codex.wordpress.org.

## Структура темы WordPress

В папке themes создаем папку и называем ее, например, mytheme. Затем в нее переносим наш style.css и создаем необходимые файлы.

**Создаем шаблоны блоков, которые отображаются на всех страницах:**

* header.php – шапка вашего сайта;
* sidebar.php – боковая колонка с различными виджитами;
* footer.php – подвал сайта.

**Шаблоны контента:**

* Index.php – шаблон главной страницы, выводятся короткие анонсы статей;
* single.php – шаблон постов (статей);
* page.php – шаблон статических страниц;
* category.php – шаблон категорий;
* search.php – шаблон выдачи результатов поиска;
* 404.php – шаблон сообщения о несуществующей странице;
* comments.php – шаблон коментариев.

**Дополнительные файлы:**

* functions.php – файл дополнительного функционала и изменения стандартного функционала движка  WordPress;
* searchform.php – код формы поиска;
* screenshot.png – Изображение 880 х 660, превью отображается в админке во вкладке «темы».

**Не обязательные шаблоны (по моему мнению):**

* author.php – шаблон страницы автора;
* archive.php – шаблон архивов;
* date.php – шаблон вывода дат;
* tag.php – шаблон меток;
* attachment.php – шаблон страниц вложения.

Лично я редко использую пять последних шаблонов, в основном делаю перенаправление на «главную» или на «404 ошибку» при запросе данных шаблонов. Информацию об авторе можно оформить на основе page.php, а остальные четыре только плодят дублированный контент на вашем сайте, организацию архивов можно реализовать намного продуктивнее.

Начинаем с файла style.css, делаем отступ и в самом верху вставляем закомментированный текст с названием и описанием темы.

/\*

Theme Name: Название темы

Theme URI: УРЛ адрес, если есть отдельный сайт о теме

Description: Описание темы оформления

Version: Версия, например (1.0)

Author: Ваше имя

Author URI: УРЛ адрес автора темы

\*/

***Файл стилей придется редактировать при подключении функционала WordPress, который добавляет дополнительные участки кода.***

### Формируем header.php

Из HTML файла переносим сюда все, что находится в <head> и элементы в <body>, которые относятся к шапке сайта. У меня в шапке большое изображение с информацией о сайте и меню.

Код был таким.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Мой блог</title>

<link href="style.css" rel="stylesheet" type="text/css" />

</head>

<body>

<div id="header">

<div id="logo">

<a href="index.html"><img src="images/shapka.png" alt="" title=""/></a>

</div>

<ul id="menu">

<li><a href="#">Главная</a></li>

<li><a href="#">Контакты</a></li>

<li><a href="#">Рубрика 1</a></li>

<li><a href="#">Рубрика 2</a></li>

</ul>

</div>

После переноса и редактирования стал таким.

<!DOCTYPE html>

<html>

<head>

<meta charset="**<?php** bloginfo( 'charset' ); **?>**" />

<title>**<?php** echo wp\_get\_document\_title(); **?>**</title>

<link href="**<?php** bloginfo('stylesheet\_url'); **?>**" rel="stylesheet" type="text/css" />

**<?php** wp\_head(); **?>**

</head>

<body>

<div id="logo">

<a href="**<?php** echo home\_url(); **?>**"><img src="**<?php** bloginfo('template\_url'); **?>**/images/shapka.png" alt="" title="" /></a

</div>

**<?php**

$args = array(

'theme\_location' => 'main-menu',

'menu' => 'main-menu',

'container' => 'nav',

'container\_class' => 'menu-{menu-slug}-container',

'container\_id' => 'main-menu',

'menu\_class' => '',

'menu\_id' => 'main-menu',

'echo' => true,

'fallback\_cb' => '\_\_return\_empty\_string',

'before' => '',

'after' => '',

'link\_before' => '',

'link\_after' => '',

'items\_wrap' => '<ul id = "%1$s" class = "%2$s">%3$s</ul>',

'depth' => 0,

'walker' => ''

);

wp\_nav\_menu( $args );

**?>**

Разбираем изменения:

* bloginfo('charset'); – устанавливает кодировку;
* wp\_get\_document\_title(); – возвращает title документа;
* wp\_head(); – это так называемый хук-действие, обязательная строка в теме;
* echo home\_url(); — выводим ссылку на главную страницу;
* bloginfo('template\_url'); – указывает путь  к папке с темой;
* bloginfo('stylesheet\_url'); – подключает файл стилей.

Нижний участок кода — это вывод меню, который вы будете настраивать из админки после подключения в functions.php, о котором написано ниже. Его нужно будет стилизовать в CSS.

Файл стилей можно подключить таким образом, но если у вас много файлов стилей и скриптов, то я рекомендую подключать их в functions.php. Подробно об этом я расписал в статье «[**Правильное подключение скриптов и стилей в WordPress**](https://zyubin.ru/cms/wordpress/beskonfliktnoe-podklyuchenie-skriptov-i-stilej-v-wordpress.html)».

***Обратите внимание, все функции WordPress заключены в конструкцию PHP!***

### Формируем index.php

Шапку мы сформировали, но теперь ее нужно подключить к нашему шаблону главной страницы, для этого в самом верху прописываем вот такую функцию.

**<?php** get\_header(); **?>**

Переносим нашу контентную часть из подготовленного HTML файла, в HTML шаблоне выводится анонс статьи с заголовком и миниатюрой, редактируем и получаем следующее.

**<?php** get\_header(); **?>**

<div class="content-main">

<div class="content">

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**

<div class="post-content">

<div class="post-img"><a href="**<?php** the\_permalink(); **?>**">**<?php** the\_post\_thumbnail('full'); **?>**</a></div>

<h2 class="post-title"><a href="**<?php** the\_permalink(); **?>**">**<?php** the\_title(); **?>**</a></h2>

<div class="post-text">**<?php** the\_excerpt(); **?>**</div>

<div class="readmore"><a href="**<?php** the\_permalink(); **?>**">Читать далее</a></div>

</div>

**<?php** endwhile; **?>**

**<?php** else: **?>**

**<?php** endif; **?>**

</div>

**<?php** get\_sidebar(); **?>**

</div>

**<?php** get\_footer(); **?>**

* the\_title(); – выводит название поста;
* the\_permalink(); – выводит ссылку на отображаемый пост;
* the\_excerpt(); – выводит установленное количество символов поста со вставкой […];
* the\_post\_thumbnail('full'); – выводит миниатюру поста, а 'full' задает полный размер;
* get\_header(); – подключение файла header.php;
* get\_sidebar(); – подключение файла sidebar.php;
* get\_footer(); – подключение файла footer.php.

***Важно. Весь наш код и функции входят в так называемый цикл WordPress, который и позволяет нам выводить несколько таких анонсов статей на одной странице, написав при этом код всего для одного анонса. Некоторые функции корректно работают только в цикле.***

Вот как выглядит чистый цикл без нашего кода.

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**<!-- Начало цикла -->

<!-- Весь код анонса поста -->

**<?php** endwhile; **?>**<!-- Окончание цикла -->

<!-- Постраничная навигация -->

**<?php** else: **?>**

<!-- Альтернативный код если нечего не найдено -->

**<?php** endif; **?>**

Все расписано, думаю с этим понятно.

### Прописываем функции в functions.php

Открываем файл и первым делом пишем конструкцию PHP, а затем в нее добавляем следующие функции.

**<?php**

/\*\*

\* Поддержка миниатюр

\*\*/

add\_theme\_support( 'post-thumbnails' );

/\*\*

\* Подключение меню

\*\*/

register\_nav\_menus( array(

'main-menu' => \_\_( 'main-menu' )

));

/\*\*

\* Подключение сайдбара

\*\*/

register\_sidebar( array(

'name' => \_\_( 'Sidebar', 'mytheme' ),

'id' => 'sidebar',

'description' => 'Правая колонка сайта.',

'before\_widget' => '<aside class="widget">',

'after\_widget' => '</aside>',

'before\_title' => '<div class="widget-title">',

'after\_title' => '</div>',

));

/\*\*

\* Функции вывода анонса поста (цитаты) на главной

\*\*/

function new\_excerpt\_length( $length ) {

return 100;//количество выводимых символов анонса (по умолчанию 55)

}

add\_filter( 'excerpt\_length', 'new\_excerpt\_length' );

/\*\*

\* Функции заменяет «[…]» на «...» в конце цитаты

\*\*/

function new\_excerpt\_more( $more ) {

return '...';

}

add\_filter( 'excerpt\_more', 'new\_excerpt\_more' );

**?>**

### Заполняем файлы sidebar.php и footer.php

В файл sidebar.php прописываем:

<div class="sidebar-right">

**<?php** if ( is\_active\_sidebar( 'sidebar' ) ) : **?>**

<ul id="sidebar">

**<?php** dynamic\_sidebar( 'sidebar' ); **?>**

</ul>

**<?php** endif; **?>**

</div>

* is\_active\_sidebar(); – проверяет, установлен хотя бы один виджет в нашей панели 'sidebar';
* dynamic\_sidebar(); – выводит активную панель виджетов.

В файл footer.php вставляем нужный вам код, например, параграф с копирайтом, а в конце перед закрывающим тегом </body> вставляем wp\_footer();. Как и wp\_head(); это обязательный хук-действие темы.

<!--- Нужный вам код --->

**<?php** wp\_footer(); **?>**

</body>

</html>

После этих настроек главная страница уже готова. Дальнейшие действия будут гораздо проще, просто будем брать уже готовый код, вставлять его в другие шаблоны и редактировать.

### Формируем page.php и single.php

Копируем код из индексного файла и вставляем в page.php, убираем лишнее и в блоке с классом .post-text изменяем вывод анонса на вывод всего контента страницы функцией the\_content();.

**<?php** get\_header(); **?>**

<div class="content-main">

<div class="content">

<div id="page">

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**

<h1 class="post-title">**<?php** the\_title(); **?>**</h1>

<div class="post-text">**<?php** the\_content(); **?>**</div>

**<?php** endwhile; **?>**

**<?php** else: **?>**

**<?php** endif; **?>**

</div>

</div>

**<?php** get\_sidebar(); **?>**

</div>

**<?php** get\_footer(); **?>**

В single.php получается тот же код, что и в шаблоне страниц, добавляем только подключение комментариев comments\_template(); из файла comments.php.

**<?php** get\_header(); **?>**

<div class="content-main">

<div class="content">

<div id="page">

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**

<h1 class="post-title">**<?php** the\_title(); **?>**</h1>

<div class="post-text">**<?php** the\_content(); **?>**</div>

**<?php** comments\_template(); **?>**

**<?php** endwhile; **?>**

**<?php** else: **?>**

**<?php** endif; **?>**

</div>

</div>

**<?php** get\_sidebar(); **?>**

</div>

**<?php** get\_footer(); **?>**

При желании можете добавить:

* the\_time('j.m.Y'); – выводит дату публикации поста (число, месяц, год);
* the\_author(); – выводит автора статьи;
* the\_category(); – выводит рубрику, в которую входит данный пост.

Шаблон комментариев довольно сложная задача для начинающих, поэтому я рекомендую взять готовое решение из стандартной темы WordPress в файлах  comments.php и functions.php и стилизовать внешний вид на ваш вкус.

### Заполняем оставшиеся шаблоны 404.php, category.php и search.php

В 404.php копируем содержимое page.php, удаляем лишнее, прописываем уведомление о том, что страницы не существует, можно подкрепить текст какой-нибудь картинкой.

**<?php** get\_header(); **?>**

<div class="content-main">

<div class="content">

<div id="page">

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**

<h1 class="page-title">Такой страницы не существует!</h1>

<img src="**<?php** bloginfo('template\_url') **?>**/images/404.png" />

**<?php** endwhile; **?>**

**<?php** else: **?>**

**<?php** endif; **?>**

</div>

</div>

**<?php** get\_sidebar(); **?>**

</div>

**<?php** get\_footer(); **?>**

В category.php просто копируем содержимое файла index.php и добавляем вывод заголовка категорий single\_cat\_title('Рубрика: ').

**<?php** get\_header(); **?>**

<div class="content-main">

<div class="content">

<div id="category">

<h1>**<?php** single\_cat\_title('Рубрика: ');**?>**</h1>

</div>

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**

<div class="post-content">

<div class="post-img"><a href="**<?php** the\_permalink(); **?>**">**<?php** the\_post\_thumbnail('full'); **?>**</a></div>

<h2 class="post-title"><a href="**<?php** the\_permalink(); **?>**">**<?php** the\_title(); **?>**</a></h2>

<div class="post-text">**<?php** the\_excerpt(); **?>**</div>

<div class="readmore"><a href="**<?php** the\_permalink(); **?>**">Читать далее</a></div>

</div>

**<?php** endwhile; **?>**

**<?php** else: **?>**

**<?php** endif; **?>**

</div>

**<?php** get\_sidebar(); **?>**

</div>

**<?php** get\_footer(); **?>**

В search.php аналогично копируем из индексного файла код и после цикла WordPress, там, где должен быть альтернативный код, о котором я говорил выше, вписываем сообщение о результатах поиска.

**<?php** get\_header(); **?>**

<div class="content-main">

<div class="content">

**<?php** if ( have\_posts() ) : while ( have\_posts() ) : the\_post(); **?>**

<div class="post-content">

<div class="post-img"><a href="**<?php** the\_permalink(); **?>**">**<?php** the\_post\_thumbnail('full'); **?>**</a></div>

<h2 class="post-title"><a href="**<?php** the\_permalink(); **?>**">**<?php** the\_title(); **?>**</a></h2>

<div class="post-text">**<?php** the\_excerpt(); **?>**</div>

<div class="readmore"><a href="**<?php** the\_permalink(); **?>**">Читать далее</a></div>

</div>

**<?php** endwhile; **?>**

**<?php** else: // Если ничего не найдено ?>

<div id="page">

<h1 class="post-title">По запросу ни чего не найдено!</h1>

<h4 class="post-text">Попробуйте еще раз с другими ключевыми словами.</h4>

</div>

**<?php** endif; **?>**

</div>

**<?php** get\_sidebar(); **?>**

</div>

**<?php** get\_footer(); **?>**

Код формы поиска размещаем в файле searchform.php. Берем стандартную форму и редактируем, в input добавляем параметр «**s**» для name и id, используем метод «GET» для отправки запроса на главную страницу home\_url('/') и очищаем УРЛ функцией esc\_url();.

<form role="search" method="get" id="searchform" class="search-form" action="**<?php** echo esc\_url( home\_url( '/' ) ); **?>**">

<div class="widget-search">

<input type="search" class="search-field" value="" placeholder="Искать..." name="s" id="s" class="input-search" />

<button type="submit" class="search-submit" >Поиск</button>

</div>

</form>

***Важно. После всех проделанных действий, нужно будет лезть в код и стилизовать все новые элементы, которые выдаст вам WordPress! Для этих целей я использую расширение «Firebug» для Mozilla Firefox, это расширение облегчает и ускоряет решение таких задач.***

На этом создание простенькой блоговой темы оформления WordPress закончено

**Подключение плагинов WordPress**

Цель познакомится с плагинами в WordPress

**Теоретические сведенья**

Плагины это один из мощнейших инструментов WordPress. По существу, плагины это модули, которые вы подключаете к своему сайту для того, чтобы активировать новые функции или элементы. Функционал, который можно добавить, зависит от назначения плагина. Существует масса плагинов, от простых (добавляют новые стили или вносят небольшие изменения в тему) до сложных (значительные изменения, такие как eCommerce интеграция или соединение со сторонними источниками). Плагины отличаются от тем и работают независимо при помощи хуков, фильтров, шорткодов, виджетов и пользовательского кода.

### **Плюсы, минусы и рекомендации**

Плагины – отличный инструмент, разработчики и администраторы сайтов получают расширенный и повторно используемый функционал. Несмотря на всю красоту, у плагинов есть свои сильные и слабые стороны. Плюсы:

Плагин можно добавить почти в любую тему, и он будет работать правильно. Плагины были разработаны, как модули, т.е. их можно копировать в новые проекты (или распространять их), что замечательно.

Плагины работают, как контейнеры для всего вашего кода и элементов. Если бы вам понадобилось разработать в теме схожий функционал, то он смешался бы с основным кодом. В плагинах же весь функционал находится в отдельном пакете.

Плагины легко создавать, а также можно воспользоваться хуками и экшенами из WordPress.

Плагины можно создавать по принципу ООП, что делает код более надежным и управляемым.

Минусы:

Ограниченный доступ к темам сайта. Плагин не может вставить код напрямую в любое место в теме. В теме должны быть специальные места, в которые можно вставить плагин, такие как место для основного контента или под виджеты.

Бывают конфликты плагинов. Иногда другие плагины (или вообще тема) могут негативно влиять на плагин, что может полностью разрушить пользовательский опыт от сайта (к примеру, вы использовали WordPress медиа загрузчик, который обновился и перестал поддерживать старый функционал, разрушая ваш плагин).

Ограниченный доступ к шаблонам и дизайнам. Плагины обязаны обеспечивать функционал, а темы – стили.

### **Основные понятия – Экшены, Фильтры, Шорткоды, Виджеты и т.д.**

Давайте выделим немного времени и обсудим ключевые понятия в разработке плагинов. Вы, должно быть, уже знакомы с этими понятиями, если вы работали с темами WordPress. Однако четкое понимание принципов работы поможет вам создавать простой в использовании и легкий в обслуживании функционал.

### **Экшены (Хуки)**

Action хук – определенный набор действий, выполняемый WordPress в определенный момент времени. При запуске экшена будут найдены все прикрепленные к нему функции, функции будут выполнены в необходимом порядке. В WordPress существуют десятки экшенов, охватывающие основной функционал. Каждому экшену присвоено уникальное имя. К примеру, когда WordPress сохраняет пост (это может быть пост, страница или другой тип контента), вызывается save\_post экшен, который ищет все прикрепленные к нему функции.

Все экшены задаются функцией do\_action(). Функция принимает следующие параметры:

$tag (обязателен)

$args (необязателен, одна или более переменных)

В общем, каждому экшену будет присвоено название (тег) и необязательное количество дополнительных полезных переменных (это могут быть множество переменных или ноль).

### **Практическое задание:**

### **Простой WordPress экшен**

Давайте рассмотрим хук save\_post. Хук принимает четыре значения. Первое – имя экшена, остальные три являются дополнительными (в которых хранятся ID поста, сам объект поста и существование или обновление поста)

// save\_post action hook

do\_action('save\_post', $post\_ID, $post, $update);

Экшен можно настроить таким образом, что при сохранении поста будет выполняться какой-либо еще функционал, такой как отправка email или обновление базы данных.

### **Создание собственного экшена**

В WordPress есть множество готовых экшенов, но вы можете создать свой. Свои экшены помогут вам вызывать свой собственный функционал точечно в любом месте темы или плагина. Воспользуемся функцией do\_action() для создания своего нового экшена. Мы будем создавать экшен, который будет вызываться сразу после запроса title на страницах.

//создаем новый экшен, который будет вызываться после запроса title

do\_action('page\_subtitle', $post);

Сначала будет отображен page, будет показан title как обычно, и только потом будет вызван наш хук page\_subtitle. WordPress проверит все прикрепленные к экшену функции и выполнит их (будет передана глобальная переменная $post, чтобы функция знала, подзаголовок какой страницы показывать).

### **Выполнение своих функций по вызову экшена**

Для вызова определенной функции во время работы экшена необходимо использовать add\_action() для прикрепления функции. add\_action принимает следующие параметры.

$hook (обязательный)

$function\_to\_add (обязательный)

$priority (опциональный)

$accepted\_args(опциональный, но обязателен если в экшен передается более одной переменной)

### **Используем обязательные аргументы**

Функцию add\_action можно вызвать, передав ей в качестве значений имя хука и вызываемой функции. Это простейший способ вызова экшена.

//когда 'save\_post' работал, вызываем функцию сохранения

add\_action('save\_post','save\_my\_custom\_post');

//Создаем нашу функцию сохранения

function save\_my\_custom\_post($post\_id){

    //код}

Как видно, при появлении save\_post мы сразу же прикрепляем нашу функцию save\_my\_custom\_post, тем самым получая возможность сделать что-то после сохранения.

### **Используем все аргументы (Понимание приоритета и значений accepted\_args)**

Иногда обязательных значений недостаточно. В таких случаях для работы экшена вам необходимо вручную задать значения переменных $priority и $accepted\_args.

Приоритет в экшене определяет порядок выполнения функций. По умолчанию экшены используют значение 10 в качестве переменной $priority. Можно задать как большее, так и меньшее значение, чтобы запускать свою функцию раньше или позже. Очень полезна, когда ваша тема или плагин используют тот же хук (вы можете задать, когда функцию следует запустить).

Переменная accepted\_args указывает на количество переменных, переданных при вызове add\_action. Количество аргументов по умолчанию, принимаемых экшеном, равно единице. Если же ваш экшен принимает несколько переменных, вы обязаны сообщить WordPress, сколько их будет. Рассмотрим экшен save\_post.

//экшен сохранения поста, задан в WordPress

do\_action( 'save\_post', $post\_ID, $post, $update );

//добавим свою функцию в 'save\_post' hook, добавив приоритет и args

add\_action('save\_post','save\_my\_page\_extended,10,3);

//функция выполняемая во время сохранения (нам нужны 3 переменные)

function save\_my\_page\_extended($post\_ID, $post, $update){

    //нам доступны все переменные}

Экшен save\_post содержит множество переменных, которые можно в него передать, мы должны задать приоритет и количество передаваемых переменных. Как только мы сообщили WordPress, что мы будем передавать несколько переменных, их можно добавить в нашу функцию, тем самым получая к ним доступ.

### **Обобщение**

В общем и целом, вы будете использовать функцию do\_action внутри темы или плагина для запуска кода в различных местах (задавая значения, доступные любой функции из хука). Для добавления своей функции необходимо использовать add\_action. Свой экшен можно вызвать с минимальным набором параметров (имя экшена и функции, которую вы хотите вызвать), или же вы можете добавить приоритет и количество передаваемых переменных.

### **Фильтры (хуки)**

Фильтра WordPress это хук, принимающий переменную (или их набор) и возвращающий ее обратно после определенной модификации. Фильтры часто используются для манипуляции информацией по умолчанию. В WordPress заложены десятки фильтров, созданных при помощи функции apply\_filters(). Функция принимает следующие аргументы:

$tag (обязательный)

$value (обязательный)

$var (опциональный)

Переменная $tag – название фильтра. $value – переменная или значение, которое необходимо отфильтровать. Переменная $var необязательная, но с ее помощью можно передать дополнительное значение в функцию. Плагины и темы часто создают фильтры таким образом, что вы можете изменять элементы без необходимости вносить изменения в код. Фильтры часто заданы внутри функций.

### **Простой фильтр WordPress**

get\_the\_excerpt – фильтр, который можно использовать внутри цикла постов для получения доступа к отрывкам статьи. Фильтр является частью ядра WordPress, задается только название фильтра и название функции. Опциональных аргументов нет.

|  |  |
| --- | --- |
| 1  2 | //получаем выдержку из поста, как задано в /wp-includes/post-template.php  echo apply\_filters( 'the\_excerpt', get\_the\_excerpt() ); |

К фильтру the\_excerpt можно прикрепить свою функцию и манипулировать выдержкой поста перед ее возвращением (к примеру, можно обернуть текст в контейнер, изменить контент или урезать длину текста).

### **Создание своего фильтра**

Рассмотрим пример. Если мы в плагине создаем массив имен, то их можно отфильтровать при помощи хука apply\_filters. Данный хук скажет WordPress, что значения могут быть изменены любой функцией, прикрепленной к хуку.

|  |
| --- |
| //Фильтруем наш массив имен  $name\_values = apply\_filters('filter\_name\_array\_values',              array('Joanna','Peter')); |

Такой фильтр поможет другим разработчикам (или вам) динамически изменять значения. Вообще, если вы пишите функцию или какой-либо другой функционал, создание фильтров считается хорошей практикой. С помощью фильтров мы расширяем элементы.

### **Выполнение своего фильтра**

Для запуска своего фильтра необходимо вызвать функцию add\_filter(). Функция принимает следующие аргументы:

$tag (обязательный)

$function\_to\_add (обязательный)

$priority (опциональный)

$accepted\_args (опциональный, но обязательный при передаче нескольких аргументов)

Функция add\_filter работает так же, как и add\_action. Задается название вызываемого фильтра, название вызываемой функции. Посмотрим, как можно использовать данную функцию для фильтрации массива имен, созданного в предыдущем примере.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | //Изменяем сохраненные значения имен во время фильтрации  function add\_new\_names($names){      $names[] = 'Simon';      return $names;  }  add\_filter('filter\_name\_array\_values','add\_new\_names'); |

Как можно заметить, мы прикрепили функцию add\_new\_names к нашему фильтру filter\_name\_array\_values. Функция принимает всего одну переменную $names. Когда мы задали наш фильтр выше при помощи apply\_filters, мы установили значения по умолчанию Joanna и Peter для массива.

Наша функция взяла существующие значения (для простоты мы их назвали $names) и добавили новое значение Simon в конец массива.

В конце мы возвращаем все значения. Возвращать значения очень-очень важно, иначе все сломается. Причина в том, что любая прикрепленная к фильтру функция тоже получит эти значения. Если вы их не вернете, функциям не с чем будет работать.

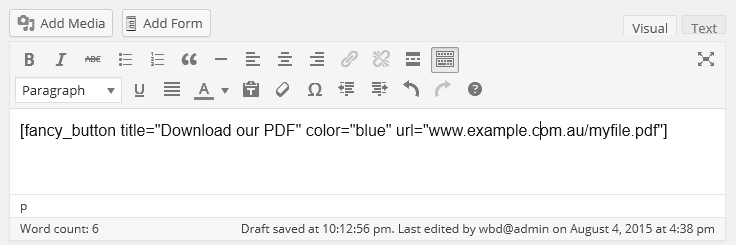
### **Обобщение**

Функция apply\_filters используется для создания фильтров внутри тем или плагинов (в них передаются значения, которые необходимо отфильтровать). Для прикрепления своей собственной функции используется add\_filter, с помощью чего можно манипулировать фильтруемым значением.

### **Шорткоды**

Так как ваши плагины, как правило, не имеют доступа к файлам темы, вам нужен способ отображать свой функционал конечному пользователю, и в то же время необходимо обеспечить легкую настройку для администратора.

Шорткоды это отличный способ решения данной проблемы. С их помощью разработчики могут создавать сложный функционал, который можно изменить при помощи простых значений и атрибутов. Шорткоды добавляются в визуальный редактор (или куда-то еще), и при просмотре их пользователями выполняется ваш код. Шорткод в редакторе:



На front-end стороне будет отображена кнопка:



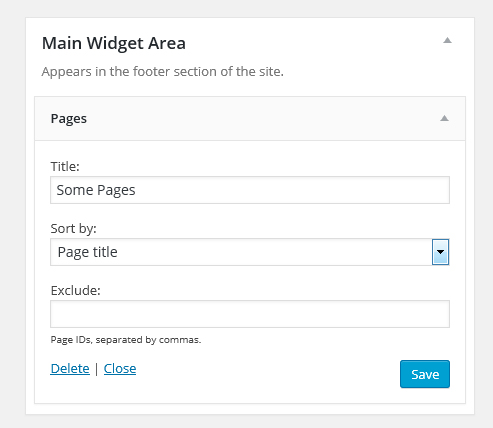
Шорткод создается при помощи функции add\_shortcode(). Первый передаваемый аргумент это название (name) шорткода, второй output\_function. Функция output\_function принимает три значения; attributes, content и name. Внутри функции создается сам шорткод и возвращается на экран. Параметры функции более подробно мы рассмотрим в нашем примере, пока что не будем обращать на них внимания. Рассмотрим простой пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //Добавляем шорткод  add\_shortcode('test\_shortcode','my\_shortcode\_output');  //выводим на экран  function my\_shortcode\_output($atts, $content = '', $tag){      $html = '';      $html .= '<p>Hello World</p>';      return $html;  } |

Теперь когда мы добавили наш шорткод [test\_shortcode] в редактор, он будет запускать нашу функцию и конвертировать результат в <p>Hello World</p>.

### **Виджеты**

Если вы хотите более подробно изучить виджеты, сначала ознакомьтесь с WordPress Widget API codex. На этой странице говорится о том, что вам необходимо, а также на данной странице вы найдете задокументированные рабочие примеры. Виджеты крайне важный инструмент, так как это еще один способ выполнения кода плагинов, и в то же время интерфейс их использования очень прост. Пример шорткода с набором конфигураций.



Так как в большинстве тем поддерживается один или несколько сайдбаров, свой виджет даст ваш быстрый способ отображения вашей информации в теме. Виджеты сложнее шорткодов, однако следуют тем же самым принципам – задаются параметры для отображения, а затем она отображаются на front-end стороне.

### **Краткое введение в виджеты**

Чтобы создать виджет, вам необходимо унаследовать класс WP\_Widget. В WordPress был реализован ООП подход к созданию виджетов, они существуют как класс; данный класс необходимо наследовать или расширить для добавления своего функционала. Рассмотрим базовый пример виджета:

//класс нашего виджета

class my\_widget extends WP\_widget{

    //конструктор (задает базовые элементы виджета)

    public function \_\_construct(){

        parent::\_\_construct(

            'my\_widget', // Base ID

            'My Widget', // Name

            array( 'description' => 'An awesome widget') // Args

        );

        add\_action('widgets\_init',array($this,'register\_widget'));

    }

    //отображение на стороне front-end

    public function widget($args,$instance ){

        echo $args['before\_widget'];

        echo $instance['title'];

        echo $args['after\_widget'];

    }

    //back-end сторона

    public function form($instance){

        $title = (isset($instance['title']) ? $instance['title'] : '');

        echo '<input type="text" id="' . $this->get\_field\_id('title') . '" name="' . $this->get\_field\_name('title') . '" value="' . $title  . '"/>';

    }

    //сохранение

    public function save($new\_instance, $old\_instance){

        $instance = array();

        $title = $new\_instance['title'];

        return $instance;

    }

    //регистрация виджета в WordPress

    public function register\_widget(){

        register\_widget('my\_widget');

    }

}

$my\_widget = new my\_widget;

Данный код создаст базовый виджет, в котором можно сохранить текстовое значение и отобразить его на экране. Как видно ниже вид не самый интересный.



### **Классы (Объектно-ориентированное проектирование)**

Объектно-ориентированное проектирование (ООП) это техника разработки, в которой функционал строится вокруг концепции классов. Классы это способ хранения функций и значений внутри одного объекта, они дают вам единую точку разработки. Общая цель ООП это создание объекта, выполняющего ваш функционал.

Плагины – отличный кандидат для ООП, так как это модули, создающиеся для выполнения определенных задач на вашем веб-сайте. В плагинах обычно содержатся свои CSS и JS файлы наряду с возможными изображениями и дополнительными PHP файлами. ООП облегчит ваш процесс разработки. Наш пример виджета будет полностью объектно-ориентированным.

Контрольные вопросы

1. Что представляют собой записи в WP, как их добавлять?

2. Что представляют собой рубрики в WP, как их добавлять?

3. Что представляют собой метки в WP, как их добавлять?

4. Что представляют собой плагины в WP, для чего они используются?

5. Что представляют собой виджеты в WP, как их добавлять?

6. Чем CMS отличаются от конструкторов сайтов?

**Разработка собственного плагина в WordPress**

**Цель:** создать свой плагин и подключить его.

**Теоретические сведенья:**

**Плагин WordPress** – это автономный код, который улучшает и расширяет функциональность WordPress. Используя любую комбинацию PHP, HTML, CSS, JavaScript/jQuery или ряда других языков программирования, плагин WordPress может добавить новые характеристики к любой части вашего сайта, включая Консоль Администрирования. Вы можете изменять поведение WordPress по умолчанию или полностью удалять ненужное поведение. Плагины позволяют легко настраивать WordPress под себя и свои потребности.

Поскольку плагин WordPress это отдельный код, он не пересекается непосредственно с какой либо частью кода ядра WordPress. Плагин может быть скопирован или установлен на любую установку WordPress. Альтернативный **(и очень не рекомендуемый)** путь внесения изменений в WordPress – написание новой функции в файл WordPress **functions.php**, который находится в каталоге **/wp-includes/**или в файл **functions.php**, который является частью вашей темы. Это может привести к ряду возможных проблем.

WordPress и его темы регулярно обновляются. И пока вы используете [дочернюю тему WordPress](https://www.hostinger.com/tutorials/how-to-create-wordpres-child-theme), очередное обновление будет перезаписывать файл **functions.php**, а добавленный вами новый код будет удален и придётся его добавлять снова и снова. Ещё один неудобный момент может возникнуть, если вы напишете много функций и в одной из них будет ошибка, которую вы никак не может отследить, вам может потребоваться заменить текущий файл оригинальным, однако при этом придётся пожертвовать всеми остальными функциями. Это повлечёт за собой большое количество PHP-ошибок на сайте, потому как вызовы уже удалённых функций ещё будут осуществляться из других мест.

Плагины же никогда автоматически не перезаписываются и не удаляются при обновлении WordPress. Если в коде вашего плагина WordPress есть ошибки, вы можете просто отключить его в Админ Консоли на время исправления ошибки. В случае, если в вашем плагине серьёзная ошибка, WordPress иногда может самостоятельно автоматически отключить его, чтобы сохранить функционирование всего сайта.

## Что такое хук (hook)?

Плагины WordPress взаимодействуют с кодом ядра через так называемые **хуки** (**hooks**, от англ. hook – крючок). К ним, как к крючкам можно “цеплять” выполнение каких-либо интересных нам функций. В WordPress есть два вида хуков:

1. Action hooks или хуки-события (добавить/удалить пользовательскую функцию к некоторому событию).
2. Filter hooks или хуки-фильтры (для изменения данных, обрабатываемых функциями).

### **События и Хуки-события**

Посещение любой страницы WordPress-сайта – это вызов множества функций PHP (называемых **событиями –** **actions**). Функции привязываются к **хукам-событиям (action hooks)**. Механизм хуков-событий предоставляется WordPress. Вы можете **добавлять** свои функции к какому-либо событию из списка событий, используя механизм хуков и они будут запускаться по запуску этого события. Вы также можете **удалять**уже существующие функции из любого списка хуков-событий. Хуки-события привязаны к выполнению определённого события. Например, перед закрытием тега **</head>** на любой странице, вызывается хук-событие **wp\_head()** и запускается ряд функций, которые привязаны к этому хуку **wp\_head()**.

Хуки-события являются контекстными – некоторые вызываются на каждой странице сайта, другие только, когда отображается Админ Консоль и так далее. Полный список событий и контекста, в котором их можно вызывать смотрите здесь [странице API плагина/Справка по Action](https://codex.wordpress.org/API_%D0%BF%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD%D0%B0/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BA%D0%B0_%D0%BF%D0%BE_Action).

#### ****Добавление функций на хук события, используя add\_action()****

Чтобы **добавить** функцию на любой хук-событие, ваш плагин WordPress должен вызвать функцию WordPress под названием **add\_action()**, как минимум с двумя параметрами.

// Хук события 'init', вызывается после того, как WordPress завершает загрузку кода ядра

add\_action( 'init', 'add\_Cookie' );

// Установка cookie с текущим временем дня

function add\_Cookie() {

setcookie("last\_visit\_time", date("r"), time()+60\*60\*24\*30, "/");

}

* Первый обязательный параметр – это название **хука-события (action hook)**, к которому собираемся связать свою функцию.
* Второй обязательный параметр – имя **функции (function)**, которую будем запускать.
* Третий параметр (необязательный) – это **приоритет** **(priority)**функции, которую вы собираетесь запустить. Вы можете привязать сколько угодно функций к одном хуку события и обращаться к ним в любом порядке. Приоритет запуска по умолчанию – 10, ваша функция запускается сразу после встроенных функций WordPress. Функция с приоритетом 11 запустится следующей и так далее.
* Четвёртый параметр (необязательный) – это **количество аргументов**, означает сколько параметров ваша пользовательская функция может принять. По умолчанию количество параметров равно 1.

#### ****Пример кода плагина для отображения текста после раздела футера (footer) на каждой странице****

Этот плагин WordPress перехватывает хук-событие **wp\_footer()**, который вызывается сразу перед закрытием тега **</body>** на каждой странице и добавляет новую функцию под названием **mfp\_Add\_Text()**. Так как плагин WordPress не привязывается к теме, то функционал будет работать даже при смене темы на другую. Можете сохранить себе этот пример в файл PHP, загрузить его в каталог **wp-content/plugins/** и активировать в Админ Консоли, чтобы увидеть изменения.

<?php

/\*

Plugin Name: Add Text To Footer

\*/

// Хук события 'wp\_footer', добавляем функцию 'mfp\_Add\_Text' к нему

add\_action("wp\_footer", "mfp\_Add\_Text");

// Определяем 'mfp\_Add\_Text'

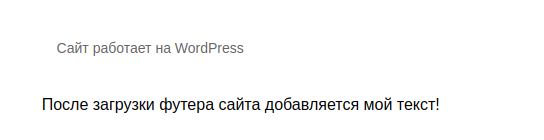
function mfp\_Add\_Text()

{

echo "<p style='color: black; padding-left: 15%;'>После загрузки футера сайта добавляется мой текст!</p>";

}

?>

Вот результат:

**Примечание:**PHP загружает содержимое сразу всего скрипта перед его выполнением, это значит, что вы можете вызвать **add\_action()** перед определением вашей функции: вызвать **add\_action()** с вашими функциями вначале файла в том порядке, в котором они будут загружаться, потом определить ваши функции в таком же порядке ниже. Такое структурирование кода сделает файл более лёгким для чтения.

#### ****Удаление функций из хуков-событий, используя remove\_action()****

Для **удаления**функции, связанной с хуком-событием, нужно воспользоваться **remove\_action()**. Пример ниже поможет понять принцип работы добавления и удаления функций в/из хука событий.

// Хук на событие 'init', который вызывается после завершения загрузки ядра WordPress и функция 'remove\_My\_Meta\_Tags'

add\_action( 'init', 'remove\_My\_Meta\_Tags' );

// Удаляем функцию 'add\_My\_Meta\_Tags' из хука-события wp\_head

function remove\_My\_Meta\_Tags()

{

remove\_action( 'wp\_head', 'add\_My\_Meta\_Tags');

}

Запускается **remove\_action()** как минимум в двумя **параметрами.**

* Первый обязательный параметр – название **хука-события (action hook)**, к которому функция была прикреплена
* Второй обязательный параметр – это имя **функции (function)**, которую вы хотите удалить
* Третий (необязательный) параметр – это приоритет удаляемой функции. Этот параметр должен совпадать с тем приоритетом, который был задан при добавлении события к хуку событий. Если он не был определён, не включайте этот параметр.

**Пример:** представим, что вы хотите, чтобы текст, добавленный в футере в предыдущем примере, не выводился по понедельникам. Один из путей реализации этого, использовать функцию даты в PHP, чтобы получить текущий день недели, проходя условным оператором **if…then…** и проверяя соответствует ли текущий день понедельнику, затем вызвать **remove\_action(),**если это понедельник. Вызываем функцию проверки и удаления по дню недели при помощи **add\_action()**, прикрепляясь к событию, происходящему раньше (например, **wp\_head**), чем то, к которому прикреплено наше основное действие выведения текста в футере (**wp\_footer)**. Если мы выберем событие происходящее позже, чем **wp\_footer**, удаления не произойдёт, поскольку страница уже будет сгенерирована.

<?php

// Хук события 'wp\_footer', запуск функции 'mfp\_Add\_Text()'

add\_action("wp\_footer", "mfp\_Add\_Text");

// Хук события 'wp\_head', запуск функции 'mfp\_Remove\_Text()'

add\_action("wp\_head", "mfp\_Remove\_Text");

// Определение функции 'mfp\_Add\_Text()', выводящей текст

function mfp\_Add\_Text()

{

echo "<p style='color: black; padding-left: 15%;'>После загрузки футера сайта добавляется мой текст!</p>";

}

// Определение функции 'mfp\_Remove\_Text()' удаление нашей предыдущей функции из события 'wp\_footer'

function mfp\_Remove\_Text()

{

if (date("l") === "Monday") {

// Target the 'wp\_footer' action, remove the 'mfp\_Add\_Text' function from it

remove\_action("wp\_footer", "mfp\_Add\_Text");

}

}

?>

### Фильтры и хуки-фильтры

**Функции фильтры** позволяют изменять возвращаемые другими функциями данные и должны быть прежде перехвачены при помощи **хуков-фильтров**. Хуки-фильтры несколько отличаются от хуков-событий. Однако, работают они подобно хукам-событиям – могут быть вызваны в разных точках скрипта и контекстно. Полный список хуков-фильтров и контекста вызова можно найти на [WordPress Plugin API/Filter Reference page.](https://codex.wordpress.org/Plugin_API/Filter_Reference)

#### Добавление фильтров, используя add\_filter()

Чтобы **добавить** любую функцию к хуку-фильтру, ваш плагин должен вызвать функцию WordPress под названием **add\_filter()**, с как минимум двумя параметрами.

// Перехват хука-фильтра 'the\_content' (содержимое любого поста), запуск функции 'mfp\_Fix\_Text\_Spacing'

add\_filter("the\_content", "mfp\_Fix\_Text\_Spacing");

// Автоматическое исправление двойного пробела

function mfp\_Fix\_Text\_Spacing($the\_Post)

{

$the\_New\_Post = str\_replace(" ", " ", $the\_Post);

return $the\_New\_Post;

}

* Первый обязательный параметр – это название **хук-фильтра (filter hook)**, на который хотим повесить функцию.
* Второй параметр – это имя **функции фильтра (filter function),** которую хотим запустить.
* Третий параметр (необязательный) – это **приоритет** **(priority)**функции, которую вы хотите запустить. В хуку-фильтру можно добавить сколько угодно разных фильтров функций. Приоритет по умолчанию – 10, устанавливает очередь вашей пользовательской функции после любой встроенной функции. Функция с приоритетом 11 запустится следующей и так далее.
* Четвёртый параметр (необязательный) – это **количество аргументов**, который означает сколько параметров ваша пользовательская функция может принять. По умолчанию количество параметров равно 1.

#### ****Пример. Плагин WordPress для изменения фрагмента/выдержки записи****

В WordPress есть функция, которая извлекает отрывок (цитату) из записи, называется она **get\_the\_excerpt()**, и также является хуком-фильтром. Функция, отображающая отрывок, вызывает **get\_the\_excerpt(),**чтоб его получить, это то место, где применяется фильтр и отрывок изменяется до отображения. Этот плагин WordPress определяет функцию фильтр, которая берёт отрывок текста как входной параметр, добавляет некоторый текст вначале него и возвращает новое значение каждый раз, когда вызывается **get\_the\_excerpt()**. Так как возвращаемое функцией **get\_the\_excerpt()**значение – это фактический фрагмент текста, он автоматически вводится как параметр функции **$old\_Excerpt,**когда вызывается, используя **add\_filter().**Функция, которую вы определяете **должна вернуть новое значение**.

<?php

/\*

Plugin Name: Add Excerpt

\*/

// Перехватываем get\_the\_excerpt фильтр хук, запускаем функцию mfp\_Add\_Text\_To\_Excerpt

add\_filter("get\_the\_excerpt", "mfp\_Add\_Text\_To\_Excerpt");

// Берём отрывок, добавляем некоторый текст перед ним и возвращаем изменённый отрывок

function mfp\_Add\_Text\_To\_Excerpt($old\_Excerpt)

{

$new\_Excerpt = "<b>Excerpt: </b>" . $old\_Excerpt;

return $new\_Excerpt;

}

?>

#### Итоговый результат: Тест фрагмента цитаты wordpress ****Удаление фильтров, используя remove\_filter()****

Для удаления фильтра нужно просто вызвать функцию **remove\_filter()**. Например, добавим снова простое условие и будем удалять текст, если сегодня четверг.

// Хук-фильтр get\_the\_excerpt, запуск функции mfp\_Add\_Text\_To\_Excerpt

add\_filter("get\_the\_excerpt", "mfp\_Add\_Text\_To\_Excerpt");

// Если сегодня четверг, удалить фильтр the\_excerpt()

if (date("l") === "Thursday") {

remove\_filter("get\_the\_excerpt", "mfp\_Add\_Text\_To\_Excerpt");

}

// Взять отрывок, добавить текст и вернуть новый отрывок

function mfp\_Add\_Text\_To\_Excerpt($old\_Excerpt)

{

$new\_Excerpt = "**Excerpt:** " . $old\_Excerpt;

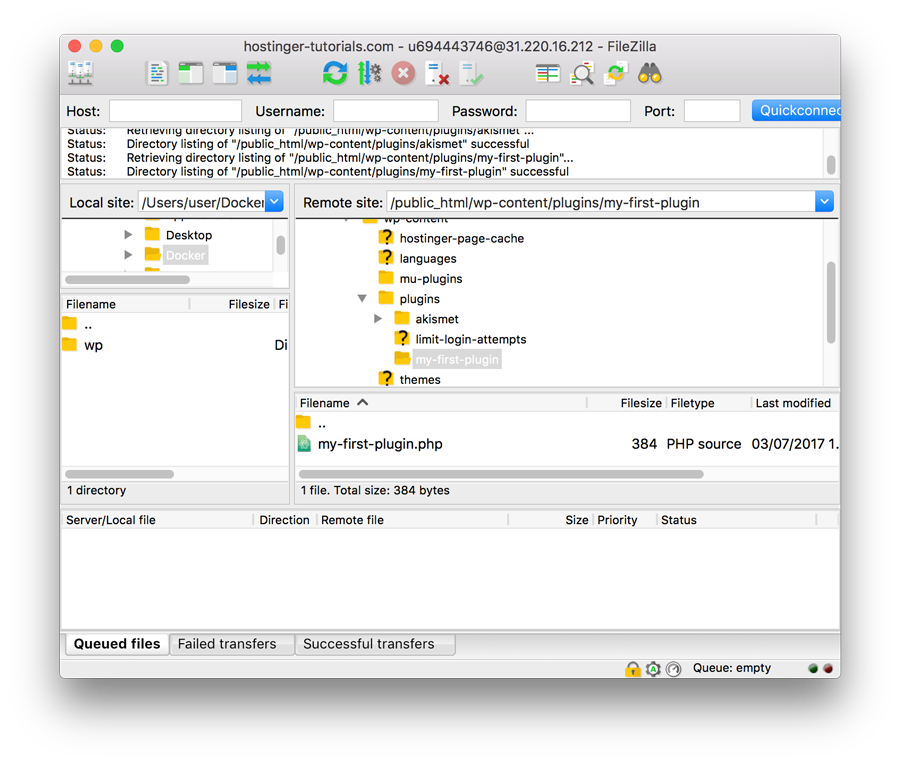
return $new\_Excerpt;

}

Теперь, когда уже есть базовое представление о хуках фильтрах и событиях, создадим простой плагин WordPress, который добавляет новую ссылку и страницу в Админ Консоль.

## **Практическое задание:**

## Шаг 1 – Сохранение плагина

Первым шагом в создании плагина для WordPress является создания каталога для хранения всех файлов. Все плагины хранятся в каталоге: /**wp-content/plugins/**. Каталог, который вы создаёте должен иметь уникальное и понятное название, не пересекающееся с другими плагинами. Поключитесь к своему аккаунту на хостинге по [FTP](https://www.hostinger.com/tutorials/ftp/filezilla-ftp-configuration). Из основного каталога WordPress перейдите в **wp-content**потом в **plugins**. Внутри каталога **plugins** создайте новый каталог с названием **my-first-plugin**.

Отдельные файлы с кодом плагина лучше тематически разделять по подкаталогам внутри основного каталога плагина вместо того, чтобы хранить всё в одном каталоге, это позволит не путаться и будет хорошим смысловым разделением исходных файлов плагина. Особенно разделение файлов по смыслу будет хорошо ощутимо, когда ваш плагин будет разрастаться и становиться более сложным. Если ваш плагин использует свои стили [CSS](https://www.hostinger.com/tutorials/css-cheat-sheet) создайте каталог **CSS** и сохраните туда все файлы CSS. Если ваш плагин WordPress использует JavaScript создайте каталог **JavaScript**.

## Шаг 2 – Создание первого файла

Первый файл вашего плагина – очень важный. Он содержит всю информацию, необходимую для WordPress, чтобы отобразить ваш плагин в списке текущих активных плагинов.

В вашем каталоге **my-first-plugin**создайте новый PHP-файл и назовите его **my-first-plugin.php**. Хорошей практикой является название основного файла плагина также как и каталога. Добавьте открывающий тег PHP **<?php** в первой строке. Вам не обязательно добавлять закрывающий тег в конце файла (чтобы понять почему, почитайте [примечание на странице PHP мануала](http://php.net/basic-syntax.instruction-separation)). Этот файл будет содержать прежде всего так называемые ‘header comments’ или заголовочные комментарии с информацией, которая будет прочитана и отображена WordPress. Header comments заключается в многострочный комментарий PHP, в каждой строке определяется специальная информация, начинающаяся со специальных символов, чтобы можно было определить, к чему эта строка относится. Такая структура должна присутствовать только в первом файле и её **не нужно** повторять в остальных.

Первой строкой в комментарии в файле **my-first-plugin.php** нужно добавить имя плагина. Начинайте многострочный комментарий PHP открывая его символами **/\***во второй строке файла, сразу после тега открытия PHP. В третьей строке напишите **Plugin Name: My First Plugin.**В двух следующих отдельных строкахможно указать описание плагина и имя автора.Затем, со следующей строки закрываем блок комментариев символами **\*/**. Ваш файл теперь выглядит примерно так:

<?php

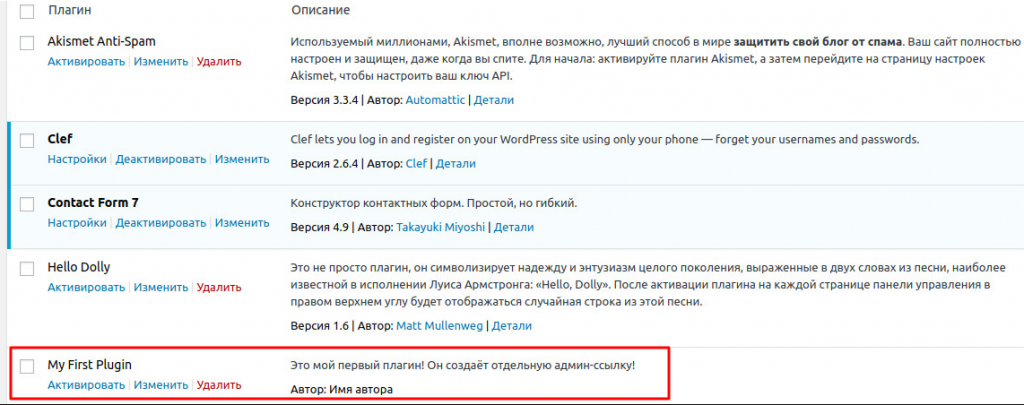
/\*

Plugin Name: My First Plugin

Description: Это мой первый плагин! Он создаёт отдельную админ-ссылку!

Author: Имя автора

\*/

Как только вы его загрузите обновите страницу **Плагины** в Админ части сайта и вы сможете увидеть в списке свой плагин, его описание и **Имя автора**.  


## Шаг 3 – Написание функций плагина

Как уже было сказано выше, хорошей практикой является разделение кода плагина в соответствующие файлы и каталоги. Так как основная функция первого файла в том, чтобы содержать в себе информацию о плагине, то сам плагин WordPress разместим в других файлах и каталогах и воспользуемся функцией PHP ‘include’ для доступа к ним. Любые файлы, хранящиеся в подкаталогах вызываются прямо из нашего кода и только из нашего кода, поэтому имя подкаталога не нуждается в префиксе. Однако, весьма рекомендуем применять уникальный префикс для ваших файлов, функций и переменных, чтобы избежать конфликтов с другими плагинами. В связи в этим мы будем использовать префикс **mfp**, сокращение от ‘My First Plugin’.

В главном каталоге вашего плагина создайте новый подкаталог с названием **includes.**Любой файл, который будет подключатся по ‘include’ в другом файле будет идти в этот подкаталог. Создайте новый файл PHP в подкаталоге **includes** и сохраните его как **mfp-functions.php.**Открываем тег **<?php** в первой строке. В этом новом файле будут храниться функции вашего плагина.

Теперь вернитесь к **my-first-plugin.php**в основном каталоге плагина. Нам нужно подключить файл **mfp-functions.php**, для того чтобы использовать новые функции. Так как это главный файл плагина, подключение **mfp-functions.php** здесь сделает функции доступными в любом другом файле вашего плагина. Используйте **require\_once**, чтобы один раз подключить файл с функциями и чтобы быть уверенным, что плагин работает только, если функции доступны. Самый простой путь подключения файлов из вашего каталога плагинов – это использование функции WordPress **plugin\_dir\_path(\_\_FILE\_\_)**, которая выдаёт полный путь к директории, где наш плагин WordPress хранится, потом использовать **.**(period), чтобы дополнить именем подкаталога, который мы создали раньше (**includes**), затем имя созданного нами файла (**mfp-functions.php**).

Редактируем **my-first-plugin.php**как показано ниже, потом сохраняем и загружаем ещё раз, перезаписывая предыдущую версию.

<?php

/\*

Plugin Name: My First Plugin

Description: Это мой первый плагин! Он создаёт отдельную админ-ссылку!

Author: Имя автора

\*/

// Подключаем mfp-functions.php, используя require\_once, чтобы остановить скрипт, если mfp-functions.php не найден

require\_once plugin\_dir\_path(\_\_FILE\_\_) . 'includes/mfp-functions.php';

Вернёмся к **mfp-functions.php**в подкаталоге **includes**.

Хорошая идея объединить несколько функций вместе и написать многострочный комментарий над каждой группой, описывая группу, затем короткий однострочный комментарий перед каждой из функций с её описанием. Таким образом вам не придётся читать весь код в поисках функции и распознаванию, что она делает. Назовём функцию **mfp\_Add\_My\_Admin\_Link().** Она будет добавлять новую верхнюю ссылку в Админ Консоли в меню навигации.

Резюмируя, пишем новую функцию следуя этим шагам:

* Пишем комментарии, описывающие функцию
* Называем функцию
* Пишем функцию

В **mfp-functions.php,** пишем следующее:

<?php

/\*

\* Добавляем новое меню в Админ Консоль \*/

// Добавляем новую ссылку в меню Админ Консоли

function mfp\_Add\_My\_Admin\_Link()

{

// My code goes here

}

Внутри нашей функции нам нужно использовать встроенную функцию WordPress **add\_menu\_page()** , чтобы задать имя меню, заголовок и определить, кто может её видеть. Затем мы говорим, что отображать, когда мы переходим на страницу. Вы можете также добавить к ссылке в меню иконку и установить её позицию в меню навигации админ панели – и то и другое по желанию, поэтому в нашем руководстве мы на этом не будем останавливаться. Рядом со ссылкой на вашу страницу будет отображаться иконка по умолчанию. Наша ссылка появится внизу меню навигации админ консоли. Вся эта информация вводится как параметры в **add\_menu\_page()**.

Четыре необходимых параметра **add\_menu\_page()** – каждый с новой строки для лучшей наглядности:

1. Название страницы, которую вы увидите после нажатия на ссылку (отображается во вкладке браузера).
2. Текст, отображаемый в ссылке меню (отображается в списке навигации Админ Консоли), это может быть название вашего плагина.
3. Требования к возможностям пользователя видеть меню, в этом примере только пользователи с возможностью ‘manage\_options’ могут иметь доступ к странице (об этой сейчас не беспокойтесь).
4. Какой файл использовать, когда открывать страницу (мы создадим его за следующим шагом), который будет храниться в подкаталоге **includes** и называться **mfp-first-acp-page.php.**URL введённый здесь известен как ‘slug’.

Прежде чем мы продолжим, важно знать, что есть альтернативный путь использования этой функции. Четвертый параметр может быть просто строкой текста, которая отображается в ссылке url после ‘**wp-admin/admin.php?page=’**. Если вы введёте ‘my-plugin-page’, URL станет **‘wp-admin/admin.php?page=my-plugin-page’**. Четвёртый параметр тогда должен быть именем функции, которая выводит что-либо. Вы можете написать функцию, которая лишь выводить ‘Welcome to page 1’, например. Значительно проще создать PHP файл, который будет содержать вашу страницу.

Редактируем **mfp-functions.php,**удаляем **// My code goes here,**заменяем его на **add\_menu\_page()** и передаём параметры, как показано ниже:

<?php

/\*

\* Добавляем новое меню в Админ Консоль

\*/

// Добавляем новую ссылку в меню Админ Консоли

function mfp\_Add\_My\_Admin\_Link()

{

add\_menu\_page(

'My First Page', // Название страниц (Title)

'My First Plugin', // Текст ссылки в меню

'manage\_options', // Требование к возможности видеть ссылку

'my-first-plugin/includes/mfp-first-acp-page.php' // 'slug' - файл отобразится по нажатию на ссылку

);

}

Чтобы запустить эту функцию нам нужно использовать функцию WordPress под названием **add\_action()** с двумя параметрами, как описано в разделе ‘Добавление функций на хук-событие’ этого руководства. Вы можете перечитать этот раздел перед тем, как продолжить.

* Первый параметр – **action hook,**который вы хотите отловить. В нашем случае хук события это **admin\_menu** – это значит, наша функция загружается тогда, когда Админ меню начинает генерироваться.
* Второй параметр – просто имя запускаемой функции. Функция, которую мы написали называется **mfp\_Add\_My\_Admin\_Link**. Обратите внимание, что круглые скобки здесь не пишутся. Помните, что PHP загружает весь скрипт до его запуска, позволяя вам использовать add\_action() раньше определения функции, указанной в параметре 2.

Наш окончательный файл выглядит так:

<?php

/\*

\* Добавляем новое меню в Админ Консоль

\*/

// Хук событие 'admin\_menu', запуск функции 'mfp\_Add\_My\_Admin\_Link()'

add\_action( 'admin\_menu', 'mfp\_Add\_My\_Admin\_Link' );

// Добавляем новую ссылку в меню Админ Консоли

function mfp\_Add\_My\_Admin\_Link()

{

add\_menu\_page(

'My First Page', // Название страниц (Title)

'My First Plugin', // Текст ссылки в меню

'manage\_options', // Требование к возможности видеть ссылку

'my-first-plugin/includes/mfp-first-acp-page.php' // 'slug' - файл отобразится по нажатию на ссылку

);

}

Загрузите файл **mfp-functions.php** в каталог **includes** и перезапишите старый.

## Шаг 4 – Создание новой админ-страницы

Теперь можно создать страницу, которая будет отображаться при нажатии на ссылку в вашей админ консоли. Вернитесь в подкаталог **includes** и создайте новый файл PHP с названием **mfp-first-acp-page.php.** Открывающий тег PHP не обязателен в этом файле, поскольку мы будем использовать только HTML. Напишите HTML-код, как показано ниже и загрузите файл.

<div class="wrap">

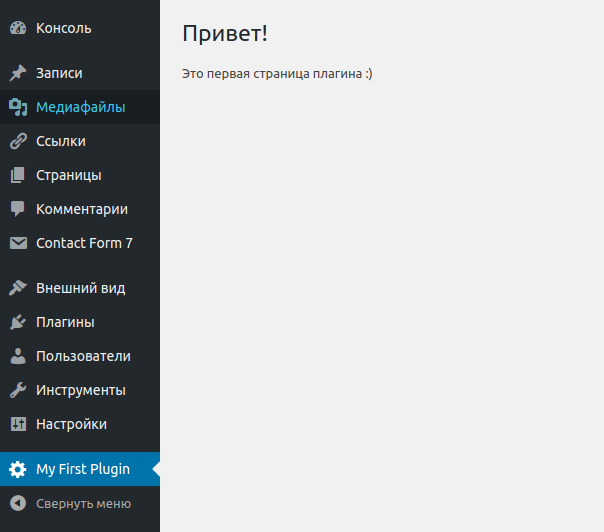
<h1>Привет!</h1>

<p>Это первая страница плагина :)</p>

</div>

При создании админ страниц, WordPress рекомендует заключать свой HTML в теги div и давать ему имя класса “wrap”, как показано выше. Это гарантирует, что весь ваш контент появится в правильном месте и будет выглядеть также как и другие админ страницы. Если не придерживаться этой рекомендации, страница может выглядеть не очень привлекательно.

Вернитесь к списку плагинов WordPress в Админ Консоли и активируйте его. Как только страница загрузится, посмотрите вниз мену навигации админ панели. Здесь вы увидите новую ссылку с именем ‘My First Plugin’. Нажмите на неё и вы увидите свою родную страницу админ управления!



## Заключение

Поздравляем, вы только что создали свой первый плагин WordPress, добавили новую ссылку в админ меню и отобразили новую страницу в Админ Консоли

**Настройка админ-панели в WordPress**

**Цель:** Настроить админ – панель на WordPress

**Теоретический материал:**

[WordPress](http://wpcustom.ru/) Admin Panel – основное место работы вебмастера. Оно должно быть комфортным, хорошо защищенным, иметь доступ ко всем необходимым функциям и приносить эстетическое удовольствие. Существует множество методов модификации консоли – разберем наиболее популярные и эффективные.

## Основные настройки

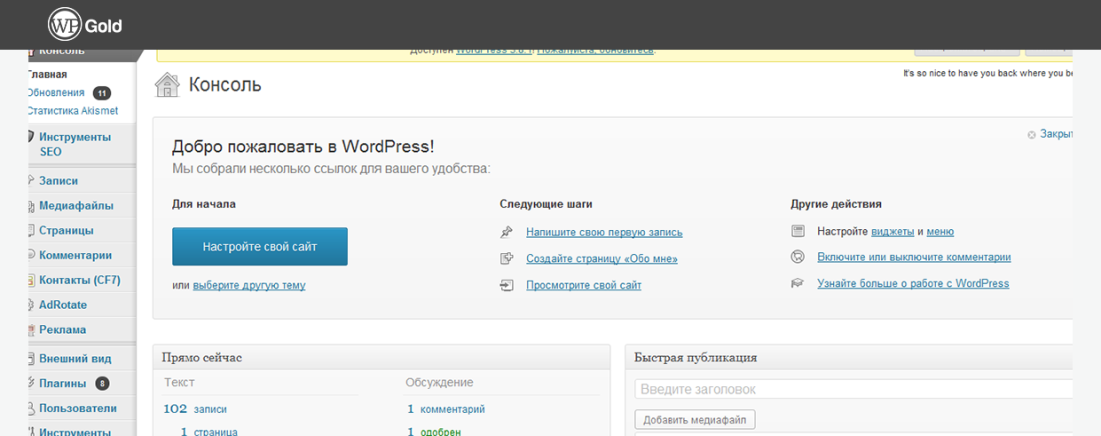
Разделяются на несколько ключевых категорий:

* улучшение безопасности – регистрация и авторизация пользователей организована достаточно просто; усилить защиту можно изменением страницы входа, добавлением дополнительных методов аутентификации;
* расширение функционала – при необходимости, в панели администратора WordPress могут быть подключены дополнительные элементы, удалены ненужные или модернизированы имеющиеся;
* внешний вид – категория изменений не считается важной, но прибавит эстетического удовольствия в процессе работы.

Это основные пункты, способствующие улучшению пользовательских факторов для вебмастера и обеспечения высокой защиты сайта. Рассмотрим их более детально.

### **Практическое задание:**

### Безопасность админ панели WordPress



Существует несколько методов обеспечения неуязвимости для консоли платформы:

* установка ограничений по IP – хорошее решение, если блог наполняется самостоятельно, и другие администраторы отсутствуют;
* капча под формой авторизации – вариант необходим для сайтов, у которых нет возможности скрыть панель регистрации; это усложнит перебор паролей и защитит от спама;
* изменение URL страницы входа – идеальный вариант для всех типов ресурсов.

При комбинировании рассмотренных методов, шансы на взлом значительно уменьшатся. Для применения этих инструментов специализированные знания не требуются – достаточно выполнить ряд несложных действий или установить соответствующий плагин.

### Ограничение по IP

Следующая манипуляция подойдет для владельцев серверов, которые работают на Apache. В файле .htaccess, находящемся в корневой папке сайта, записываются следующие строки в начале кода:

<FilesMatch  "^(wp-login|wp-config)\.php$">

  Order deny,allow

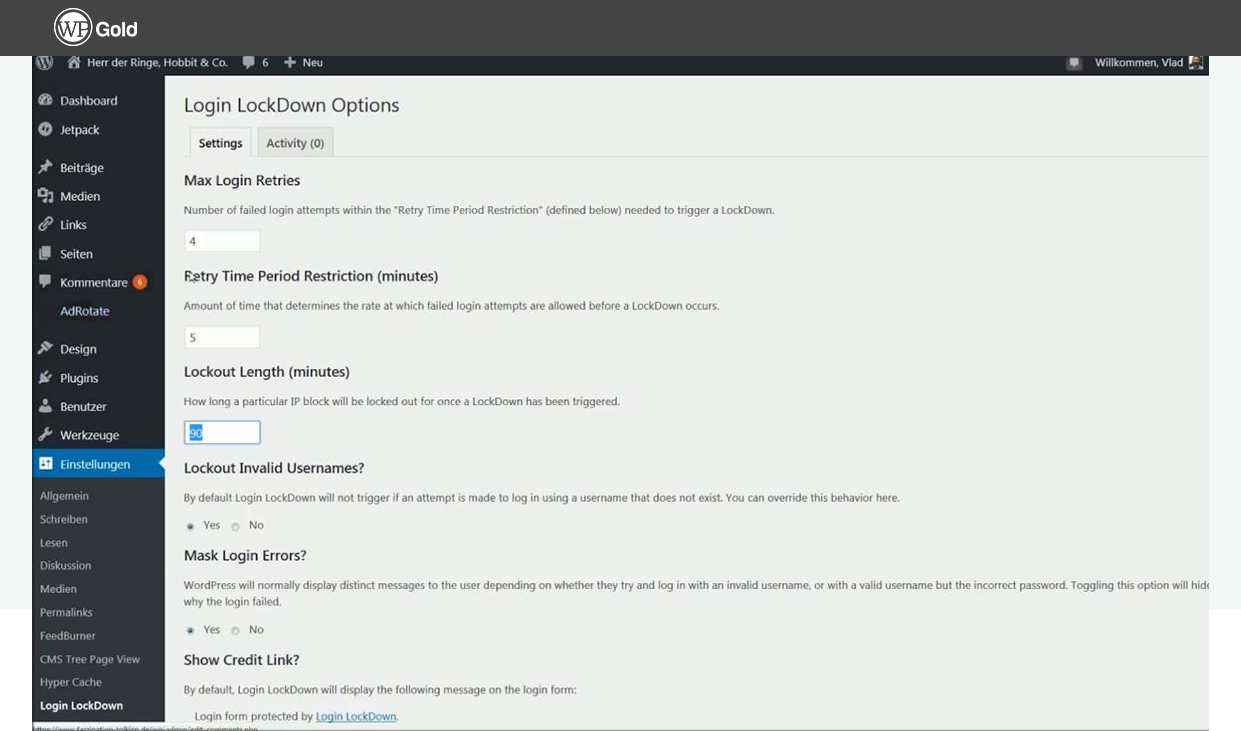
  Deny from all

  Allow from 192.168.4.6/16

</FilesMatch>

Где 192.168.4.6/16 – IP-адрес подсети, предоставляемый провайдером вебмастера. Либо указывается статичный, при использовании такового. Адрес можно узнать через сервис 2ip.ru. Скопировав предоставленные данные на сайт Who.is, система предоставит подсеть (строка CIDR).

### Защита от перебора паролей

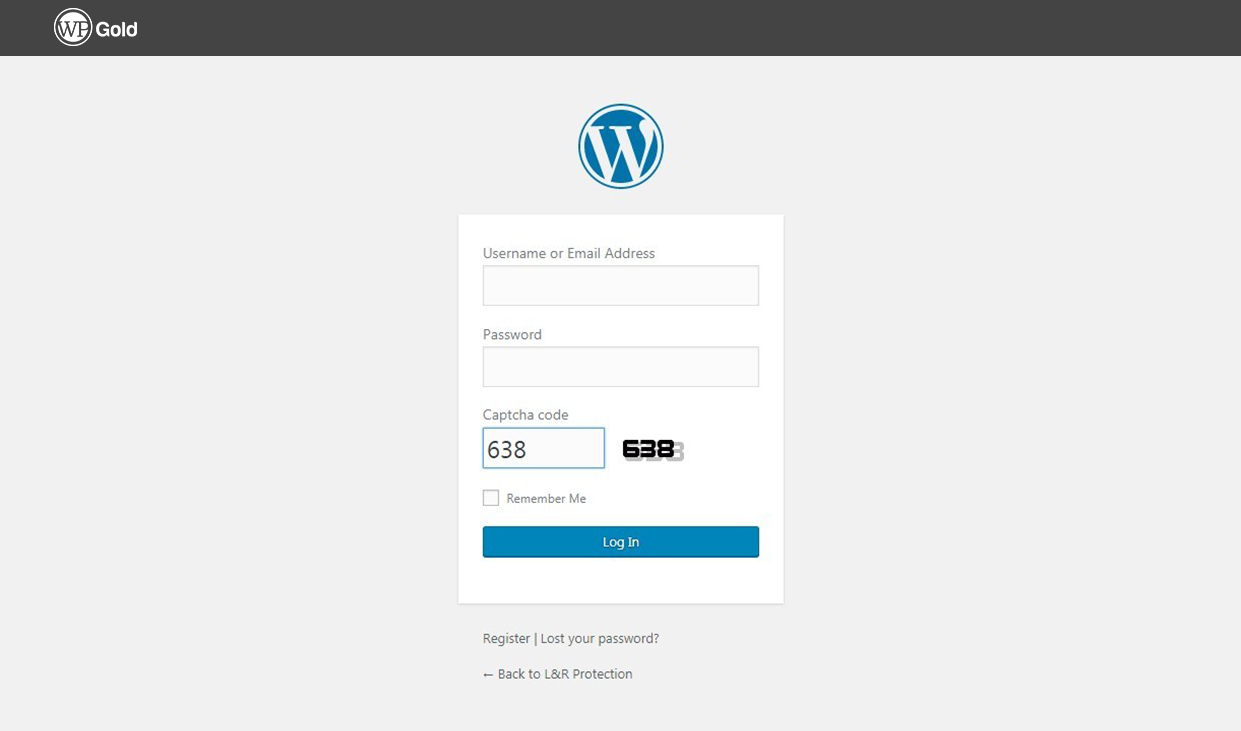


Популярным и самым эффективным методом является применение плагина Login Lockdown. При достижении определенного количества попыток авторизации – система блокирует дальнейший ввод данных и сайт ограничивает доступ для текущего IP. После установки расширения необходимо зайти в меню настроек и указать нужные параметры (условия и сроки блокировки). Окно параметров:

* Max Login Retries – установка максимального количества неудачных попыток входа;
* Retry Time Period Restriction (minutes) – время, в течение которого указанное количество неудачных попыток ввода логина и пароля приведет к бану;
* Lockout Length (minutes) – время блокирования доступа по IP;
* Lockout Invalid Usernames? – использование фильтра плагина для попыток входа под именем несуществующего пользователя;
* Mask Login Errors? – включение отображения уведомлений при неудачной авторизации;
* Show Credit Link? – указывать или нет в панели регистрации ссылку на плагин.

После этого, ниже формы авторизации панели администратора WordPress будет отображено уведомление, что ресурс защищен при помощи Login LockDown.

### Использование капчи



Подойдет для сайтов, где работает несколько администраторов или для борьбы со спам-пользователями. Достаточно установить плагин Google Captcha (reCAPTCHA) и выполнить его настройку. Чтобы расширение нормально функционировало, следует зарегистрироваться на Google Recaptcha для получения ключей активации. После сохранения данных, форма авторизации будет защищена от перебора паролей.

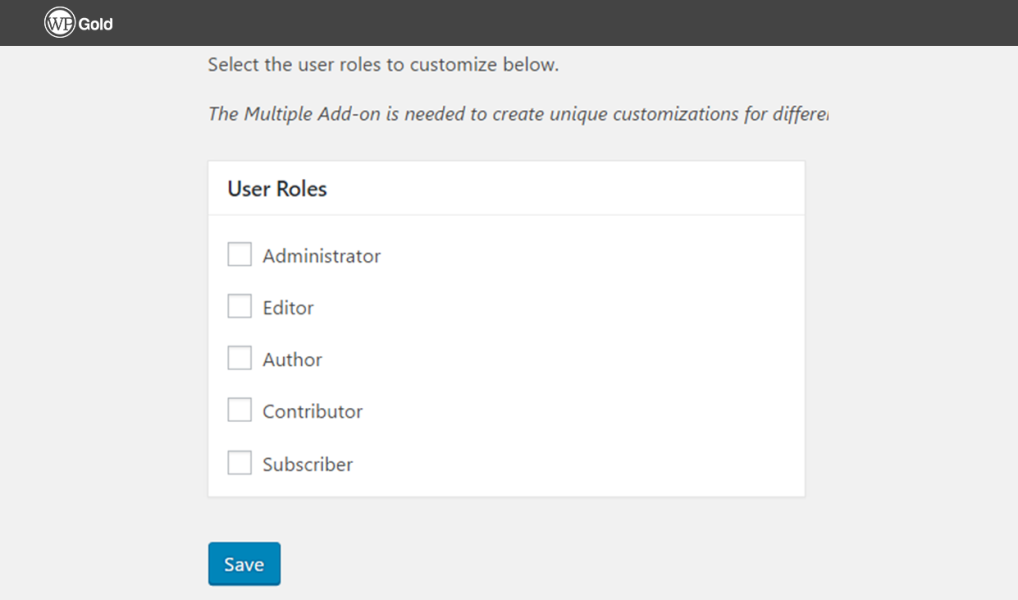
### Изменение адреса админ панель WordPress

Рекомендуется для большинства ресурсов, как один из простых и наиболее эффективных методов. Заключается в установке плагина WPS Hide Login. В настройках записывается новый адрес панели входа (его следует запомнить или записать, чтобы не пришлось восстанавливать доступ). Последующий вход осуществится по указанной в параметрах ссылке.

### Настройка и расширение функционала

Большинство дополнительных параметров спрятано от глаз пользователей в специальном выплывающем меню в верхней части сайта «Настройка экрана». Включив его, отобразится список с возможными инструментами (для каждой страницы панели необходимо отдельное редактирование). Можно подключить дополнительные блоки, изменить количество отображаемых постов, включить полноэкранный режим редактора.

### WordPress Admin Panel по ролям



Такая функция будет востребована вебмастерами, которые предоставляют пользователям доступ к консоли (в зависимости от их роли). Применив плагин WP Admin UI Customize можно создать отдельную панель для каждой группы и выставить отдельные ограничения. Предоставляется достаточно большое количество настроек в удобной рабочей панели. Для разработки уникальной панели администратора для каждой категории пользователей необходимо перейти на платную версию.

Второй плагин – WordPress Admin Menu Editor, позволит изменить панель управления сайтом. Прилагаются следующие возможности:

* редактирование, добавление и удаление пунктов в консоли, в том числе тех, что созданы темами или плагинами;
* управление ролями пользователей;
* поддержка мультисайта;
* функция drag-and-drop для редактирования меню;
* скрывание плагинов в консоли.

Достаточно продвинутое расширение, позволяющее сделать подменю в WordPress, изменить оформление консоли, скрывать ненужные элементы, выполнить отладку. При необходимости можно купить платный пакет с расширенным функционалом.

### Удаление админ-меню

Тех, кого раздражает верхняя полоса в консоли администратора и на сайте, можно избавиться от нее при помощи следующего кода:

add\_action ( 'admin\_head', 'kill\_adminmenu');

function kill\_adminmenu (){

  $GLOBALS['menu'] = array();

  echo "<style type='text/css'>#adminmenuwrap, #adminmenuback{display:none!important;} #wpcontent{margin-left:10px!important;}</style>";

}

Вставляется в файл functions.php. Для отмены изменений – скопированный код удаляется

Вариант 2 содержимое <style>  в 4 строке добавить к стилям темы в файл style.css.

### Отключение сообщений об обновлении

Пакеты с новыми версиями платформы выпускаются регулярно – они необходимы для получения нового функционала и обеспечения должного уровня безопасности. Несет исключительно информационный характер, скрывается с целью предотвращения волнений у клиентов или владельцев сайтов. Указанные строки копируются в functions.php:

if( ! current\_user\_can ( ' edit\_users' ) ){

add\_filter  ( 'auto\_update\_core', '\_\_return\_false' ); // обновление ядра

add\_filter  ( 'pre\_site\_transient\_update\_core', '\_\_return\_null' );

}

### Лишние виджеты админ панели WordPress

Новые версии платформы на основном экране устанавливают блоки с информацией о блоге, новостями и статистикой. Для некоторых вебмастеров такие виджеты не нужны и не приносят пользы – их рекомендовано отключить. В functions.php вносится код:

add\_action( 'wp\_dashboard\_setup' , 'clear\_dash', 99 );

function clear\_dash (){

  $side = & $GLOBALS['wp\_meta\_boxes']['wp\_meta\_boxes' 'dashboard']['side']['core'];

  $normal = & $GLOBALS['wp\_meta\_boxes']['wp\_meta\_boxes' 'dashboard']['normal']['core'];

  $remove = array(

   'dashboard\_activity', // последняя активность;

   'dashboard\_primary', // новости wordpress;

   //'dashboard\_right\_now', // консоль;

  );

  foreach( $remove as $id ){

   if ( isset($side[$id]) ) unset( $side[$id] );

   elseif ( isset($normal[$id]) ) unset( $normal[$id] );

  }

  // remove welcome panel

remove\_action ('welcome\_panel', 'wp\_welcome\_panel');

}

### Административные меню

Слева в консоли имеется область меню с различными блоками: параметры, обсуждения, настройки. Порой для пользователей необходимы дополнительные пункты для быстрого доступа к определенным задачам. Рассмотрим, как это сделать на примере баннеров. Функция add\_menu\_page() записывается в файл function.php:

add\_menu\_page( $page\_title, $menu\_title, $capability, $menu\_slug, $function, $icon\_url, $position );

Где:

* $page\_title – название, отображаемое в заголовке страницы настроек;
* $menu\_title – имя пункта в меню;
* $capability – настройки уровня доступа;
* $menu\_slug – URL-адрес страницы с настройками;
* $function – генерирует страницу;
* $icon\_url – адрес расположения иконки (если не указано будет отображаться шестеренка);
* $position – установка порядка (если не выбрано – помещается в конце).

Указанный код помещается в functions.php:

add\_action ('admin\_menu', 'my\_admin\_menu');

function my\_admin\_menu() {

add\_menu\_page ('Параметры каталога', 'Каталог', 1, 'catalog.php', 'print\_page\_function');

  function print\_page\_function() {

    // тут будет находиться содержимое страницы

  }

}

Функция add\_submenu\_page() включает субменю WordPress. Это необходимо для того, чтобы прикрутить пункт под основное меню (к примеру «Параметры», «Инструменты»). В данном варианте, изначальный код дополняется $parent\_slug – элемент, указывающий на ссылку родительского пункта:

add\_submenu\_page( $parent\_slug, $page\_title, $menu\_title, $capability, $menu\_slug, $function );

Пример прикрепления субменю к «Консоли»:

add\_action ('admin\_menu', 'banners\_menu');

function banners\_menu () {

add\_submenu\_page  ('index.php', 'Настройка баннеров', 'Баннеры', 1, 'banners-status.php', 'banner\_on\_off');

  function function banner\_on\_off () {

    // тут будет находиться содержимое страницы

  }

}

## Плагины для работы с панелью администратора WordPress

Расширения помогли менее опытным вебмастерам сдружиться с разработкой сайтов. Значительно уменьшилась работа с кодами и файлами конфигурации. Теперь достаточно установить специализированный плагин и несколькими кликами наладить необходимую функциональность. Рассмотрим подборку [плагинов](http://wpcustom.ru/plaginy_dlya_wordpress/), упрощающих работу с консолью.

* Admin Tweaks – настройка различных элементов в панели управления, может изменять основные и дополнительные параметры, меню и профили пользователей. Работает с колонками и списками записей.
* Adminimize – позволяет скрыть неиспользуемые пункты меню, или ограничив их показ определенной категории пользователей. Будет полезен сайтам, где трудиться множество редакторов.
* WP Total Hacks – более 20 новых функций и настроек стандартной админ панели WordPress. Может изменять виджеты, текст, таблицы, интерфейс.
* TinyMCE Advanced – обязательный к установке модуль. Значительно расширяет спектр инструментов стандартного редактора. Оформление контента станет более разнообразным.
* Options Framework – инструмент, позволяющий создать панель с управлением внешним видом и функциональностью темы. С его помощью можно добавить поля, строки, чекбоксы, выпадающие списки, изменять цвета.

Все расширения созданы для облегчения и ускорения рабочего процесса с контентной частью сайта.

## Заключение

Рассмотрены основные моменты для обеспечения безопасности, настройки функциональности и удобства пользования панелью администратора WordPress.