

????

- [Темы: введение](#)
- [Темы: разметка](#)

?????: ??????????

Два вида тем: классические (поддерживают все версии WP, php-js-html) и блочные (с версии 5.9, WYSIWYG-конструктор). Есть гибридные, но это моветон.

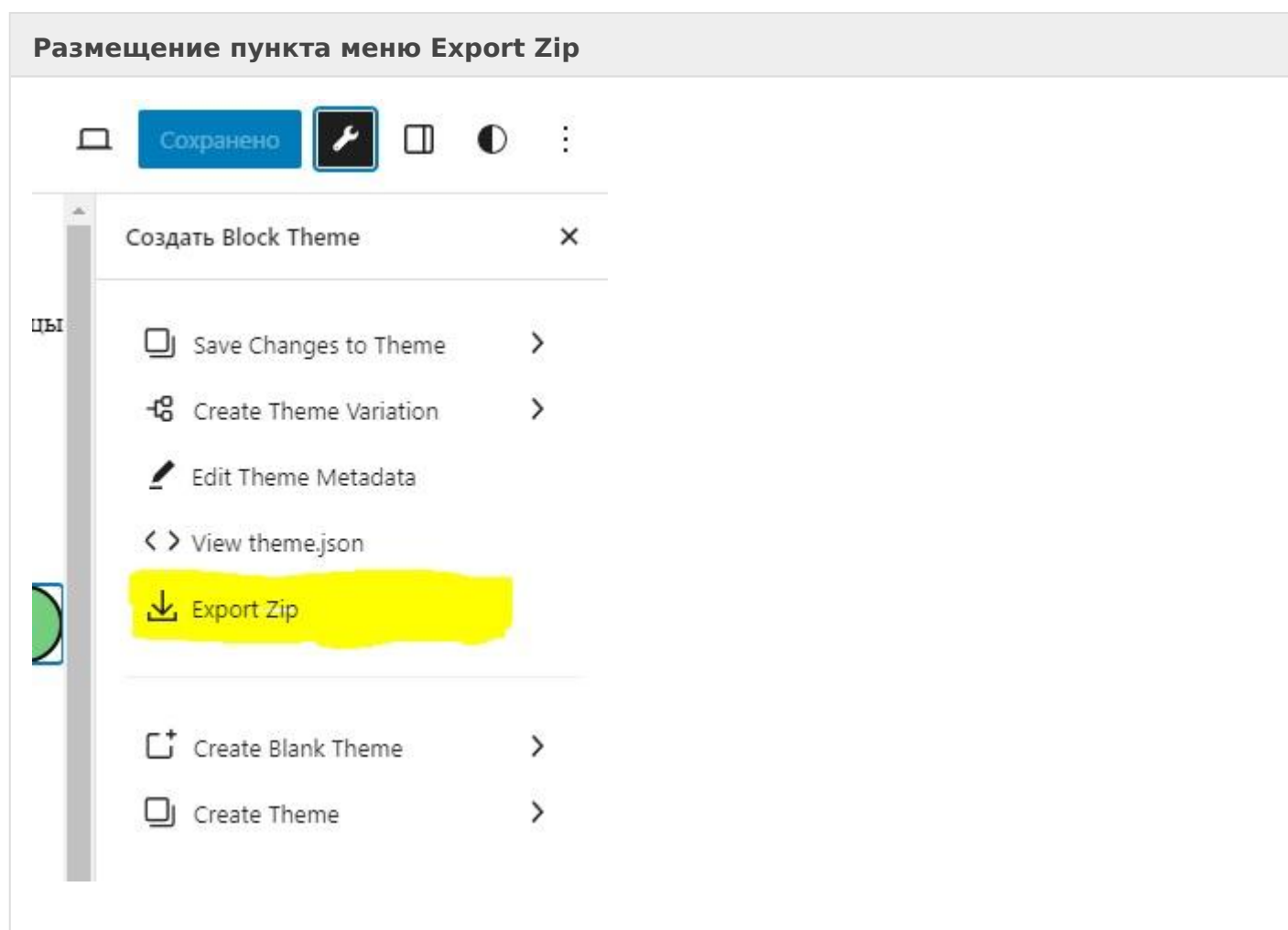
Создание новой блочной темы:

Либо модифицируя во встроенном редакторе существующую тему, либо используя плагин

Экспорт тем

Экспорт блочных тем через стандартный экспортер:

При редактировании любой страницы темы нажать Настройки - Export Zip. Тема сохранится в zip файле, который импортируется в другой проект.



Экспорт блочной темы через плагин Create Block Theme ([Плагины](#))

Очень похоже на стандартный экспорт.

Изменение имени темы

При экспорте обновленной темы сохраняется название предыдущей темы. Для изменения после экспорта нужно:

- Распаковать архив.
- Переименовать созданную папку
- В корне созданной папки открыть `style.css` и изменить параметры имени шаблона, адреса размещения шаблона, имени разработчика, адреса сайта разработчика:

```
/*
Theme Name: Twenty Twenty-Four
Theme URI: https://wordpress.org/themes/twentytwentyfour/
Author: the WordPress team
Author URI: https://wordpress.org
...
```

- Желательно добавить в корень файл `screenshot.[png|jpg]` для отображения скрина темы
- Заархивировать папку

Структура проекта шаблона

```
parts/
  footer.html
  header.html
patterns/
  example.php
styles/
  example.json
templates/
  404.html
  archive.html
  index.html (required)
  singular.html
README.txt
functions.php
screenshot.png
style.css (required)
theme.json
```

style.css

Метаданные шаблона:

Записываются между открывающих и закрывающих комментариев /* */

Параметр	Определение
Theme name:	Название темы
Theme URI:	Адрес страницы темы
Description:	Описание темы
Version:	Версия в формате X.X или X.X.X
Author:	Имя автора или организации - автора темы
Author URI:	Адрес автора
Tags:	Разделенные запятыми теги. Здесь размещены теги для размещения в официальном хранилище, но для своих тем может быть другая система.
Text Domain:	Дословно: Строка, используемая в текстовом домене для переводов. Хз что такое практически. Но данное имя используется при названии папки темы, а также префикс при инкюде CSS JS
Domain Path:	Связано с переводом.
Tested up to:	Максимальный номер версии WP, на котором тестировался шаблон
Requires at least:	Минимальная версия, на которой тестировался шаблон.
Requires PHP:	Минимальная требуемая версия PHP
License: Licence URI:	Название и адрес лицензии.
Template:	Название темы-родителя (обычно совпадает с названием папки). Используется при написании темы-потомка.

Специфичные стили шаблона

Глобально применяемые стили. Обычно для этого используют файл theme.json но иногда этого не хватает.

index.html Шаблон по-умолчанию для страниц.

theme.json

Файл настроек глобальных переменных, стилей шаблона. Структура файла:

```
{
  "$schema": "https://schemas.wp.org/trunk/theme.json",
  "version": 2,
  "settings": {},
  "styles": {},
  "customTemplates": {},
  "templateParts": {},
  "patterns": []
}
```

version: версия json-схемы у файла. Сейчас 2.

\$schema: адрес схемы, для поддержки редакторами.

settings: какие элементы управления блоками отображаются, настройки пресетов и многого другого.

styles: применения цветов, размеров шрифтов, пользовательского CSS и других стилей к веб-сайту и блокам.

customTemplates: Метаданные для пользовательских шаблонов, определенных в папке /templates вашей темы.

templateParts: Метаданные для частей шаблона, определенных в папке /parts вашей темы.

patterns: массив шаблонных элементов, которые должны быть зарегистрированы из каталога шаблонов.

Раздел styles

Поддерживаемые уровни

- Глобальный, добавляется на верхний уровень иерархии styles

```
{
  "version": 2,
  "styles": {
    "color": {
      "text": "#000000",
      "background": "#f5f1ea"
    }
  }
}
```

- Элементы

```
{
  "version": 2,
  "styles": {
```

```
    "elements": {
      "button": {
        "color": {
          "text": "#ffffff",
          "background": "#aa3f33"
        }
      }
    }
  }
}
```

Поддерживаемые элементы:

- button
- caption
- cite
- heading
- h1 - h6
- link
- Блоки

Добавление включений (assets).

Функции получения адресов:

- `get_stylesheet_uri()`: Возвращает адрес до файла `style.css` активной темы.
- `get_theme_file_uri($file)`: Возвращает URL активной темы, с необязательным параметром `$file`. Отдает URL родительской темы если в дочерней нет файла.
- `get_parent_theme_file_uri($file)`: Отдает URL родительской темы с необязательным параметром `$file`.

Внедрение общего CSS

Необходима в файл `functions.php` добавить хук на событие `wp_enqueue_scripts`. Функция, выполняющаяся при вызове, должна содержать одну или более функцию `wp_enqueue_style`

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

```
wp_enqueue_style(
    string $handle,
    string $src = '',
```

```
    []string[] $deps          = array(),
    []string|bool|null $ver = false,
    []string $media          = 'all'
);
```

\$handle имя или ID файла стиля, префикс должен быть имя темы (параметр Text Domain в настройках).

\$src URL файла стиля.

\$deps массив файлов стилей, от которых зависит включаемый стиль.

\$ver версия

\$media для какого типа данных используется этот стиль, например all (default), screen, print, handheld.

Пример:

В файл functions.php добавляем код:

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

function theme_slug_enqueue_styles() {
    []wp_enqueue_style(
    [] 'theme-slug-style',
    [] get_stylesheet_uri()
    []);
    wp_enqueue_style(
    [] 'theme-slug-primary',
    [] get_parent_theme_file_uri( 'assets/css/primary.css' )
    []);
}
```

Внедрение inline CSS (точно не догнал)

Используется функция

```
wp_add_inline_style(
[]string $handle,
[]string $data
);
```

- **\$handle** уже существующий ID файла стиля (добавленный ранее при помощи wp_enqueue_style)
- **\$data** фильтр для добавления в страницы

Внедрение JS

Аналогично стилям, только функция

```
wp_enqueue_script(  
    string $handle,  
    string $src          = '',  
    string[] $deps       = array(),  
    string|bool|null $ver = false,  
    array|bool $in_footer = false  
);
```

- **\$handle:** имя или ID файла стиля, префикс должен быть имя темы (параметр Text Domain в настройках).
- **\$src:** URL файла стиля
- **\$deps:** Массив имен скриптов, от которых зависит данный скрипт
- **\$ver:** версия
- **\$in_footer:** Определяет точку загрузки скрипта и тип загрузки (синхронных/асинхронный). Массив значений:
 - strategy: 'defer' (по умолчанию) или 'async'
 - in_footer: Логическое, false - в header

Пример:

```
add_action( 'wp_enqueue_scripts', 'theme_slug_enqueue_scripts' );  
  
function theme_slug_enqueue_scripts() {  
    wp_enqueue_script(  
        'theme-slug-navigation',  
        get_parent_theme_file_uri( 'assets/js/navigation.js' ),  
        array(),  
        wp_get_theme()->get( 'Version' ),  
        true  
    );  
}
```

Также может быть inline, editor.

Внедрение изображений

Внедрение в шаблон изображений напрямую встречается нечасто, но пример для внедрения из родительского или дочернего:

```
  

```

?????: ??????????

Каждый шаблон/блок/часть состоит из функциональных блоков.

Редактирование: Внешний вид-Редактор-Шаблоны-Нужный шаблон-Три точки справа-Редактор кода

Общая структура функционального блока:

```
<!-- wp:namespace/slug {"align":"full"} /-->
```

- **Prefix:** Префикс wp: определяет, что это не комментарий
- **Namespace:** Пространство имен блока (при использовании core блоков, пространство имен не используется)
- **Slug:** имя файла блока
- **Block Settings:** JSON для настройки блока

Например, шаблон страницы с header, footer

```
<!-- wp:template-part {"slug":"header","tagName":"header"} /-->

<!-- wp:group {"tagName":"main","layout":{"type":"constrained"}} -->
<main class="wp-block-group">
  <!-- wp:template-part {"slug":"loop","align":"full"} /-->
</main>
<!-- /wp:group -->

<!-- wp:template-part {"slug":"footer","tagName":"footer"} /-->
```

Создание нового шаблона

Создаваемый шаблон: content-canvas

- Создать пустой файл content-canvas.html в templates

```
docker exec -it id_cont bash #в случае докера
cat /dev/null > content-canvas.html
```

- Зарегистрировать шаблон в themes.json

```
{
  "customTemplates": [
    {
      "name": "content-canvas",
      "title": "Content Canvas",
      "postTypes": [
        "page",
        "post"
      ]
    }
  ]
}
```

- Редактировать в редакторе кода, например:

```
<!-- wp:template-part {"slug":"header","tagName":"header"} /-->

<!-- wp:group {"tagName":"main","layout":{"type":"default"}} -->
<main class="wp-block-group">
  <!-- wp:post-content {"layout":{"type":"constrained"}} /-->
</main>
<!-- /wp:group -->

<!-- wp:template-part {"slug":"footer","tagName":"footer"} /-->
```

Через редактор кода можно к одному элементу применить нужный класс.