

Stirling Engine Models - Movement is Everything

High-precision hot air powered Stirling Engines as Finished Models and Kits! boehm-stirling.com

Gulp

Gulp для самых маленьких - подробное руководство

09

03/2016

🔧 Инструменты

Gulp для самых маленьких - подробное руководство

Всем привет, друзья! Сегодня мы подробно рассмотрим, что такое Gulp и как с его помощью можно автоматизировать работу Front-end разработчика. В результате урока мы соберем серьезное и внушительное рабочее Front-end окружение для веб-разработки с использованием Gulp.

Видео урок:



Поделиться



Твитнуть



Поделиться 7



Класснуть



Плюсануть



Запинить

Gulp для самых маленьких - подробное руководство



СМОТРЕТЬ УРОК НА YOUTUBE





- Урок по обновлению Gulp до 4 версии: [Ознакомиться с уроком Gulp 4](#)

Основные ресурсы урока:

Node.js: <https://nodejs.org/en/>

Gulp: <http://gulpjs.com/>

Bower: <http://bower.io/>

Browsersync: <https://browsersync.io/>

Проект-пример из данного урока вы можете посмотреть на GitHub и скачать: <https://github.com/agragregra/gulp-lesson>

Gulp - это инструмент, который помогает автоматизировать рутинные задачи веб-разработки. Gulp предназначен для решения таких задач, как:

- Создание веб-сервера и автоматическая перезагрузка страницы в браузере при сохранении кода, слежение за изменениями в файлах проекта;
- Использование различных JavaScript, CSS и HTML препроцессоров (CoffeeScript, Less, Sass, Stylus, Jade и т.д.);
- Минификация CSS и JS кода, а также, оптимизация и конкатенация отдельных файлов проекта в один;
- Автоматическое создание вендорных префиксов (приставок к названию CSS свойства, которые добавляют производители браузеров для нестандартных свойств) для CSS.
- Управление файлами и папками в рамках проекта - создание, удаление, переименование;
- Запуск и контроль выполнения внешних команд операционной системы;
- Работа с изображениями - сжатие, создание спрайтов, ресайз (png, jpg, svg и др.);



- Деплой (отправка на внешний сервер) проекта по FTP, SFTP, Git и т.д.
- Подключение и использование в проекте безгранично большого количества Node.js и Gulp утилит, программ и плагинов.
- Создание различных карт проекта и автоматизация другого ручного труда.

Можно с уверенностью сказать, что Gulp и множество утилит, написанных для него, подходят для решения практически любой задачи при разработке проекта любой сложности - от небольшого сайта до крупного проекта.

Любой проект, использующий Gulp имеет в корне файл **gulpfile.js**, который содержит набор инструкций по управлению проектом. Сразу хочется сказать, что написание инструкций для Gulp не является программированием, хотя пишутся на языке JavaScript. Не стоит пугаться больших gulpfile.js, в основном все инструкции однотипные и имеют общие черты. К тому времени, как вы прочтете данное руководство, у вас не должно остаться вопросов по Gulp, так как система сборки элементарная. Но если у вас останутся вопросы - обязательно пишите в комментариях.

Установка Gulp

Для работы с Gulp у вас должен быть установлен [Node.js](#). Установка Node.js для различных платформ довольно простая - скачиваете инсталлер Node для своей операционной системы и устанавливаете. Я рекомендую устанавливать последнюю версию Stable. Для пользователей Linux я подготовил отдельное видео руководство по установке:

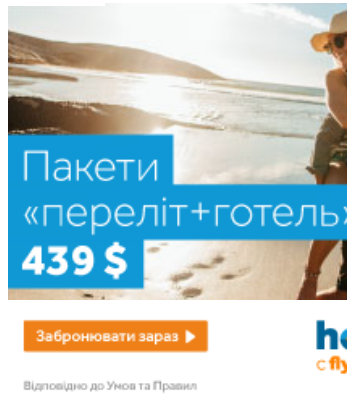
 Поделиться  Твитнуть  Поделиться 7  Класснуть  Плюсануть  Запинить

Установка Node.js + Gulp + _optimized_gulp_sass в Linux

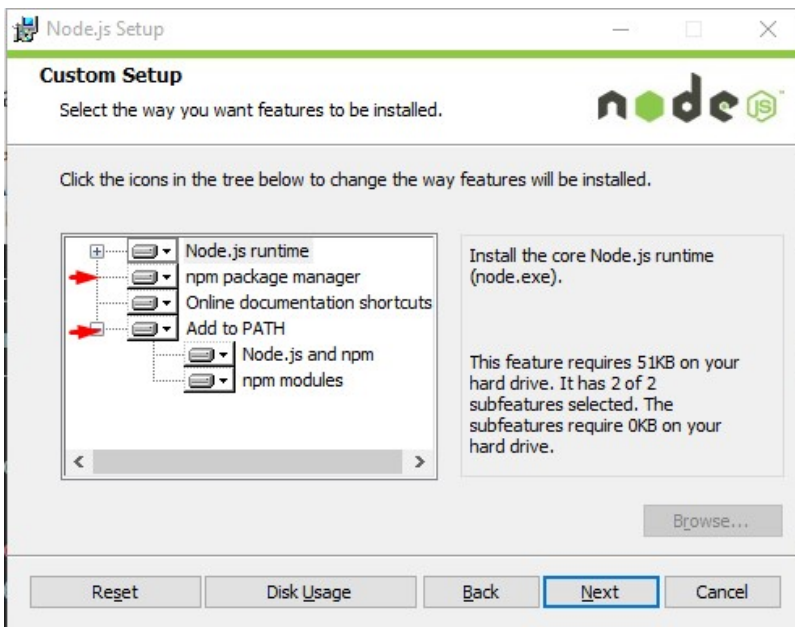


 СМОТРЕТЬ УРОК НА YOUTUBE





Обратите внимание, что при установке Node.js необходимо отметить галочками установку npm и добавление программы в Path:



После того, как Node установлен, можно приступать к установке Gulp. Откройте терминал (Командная строка в Windows) и выполните следующую команду:

```
code source
1. npm i gulp -g
```

Для пользователей Mac и Linux, возможно, придется выполнять команды с правами суперпользователя, **sudo**.

Из данной команды мы видим, что запускается менеджер пакетов **npm** (Node Package Manager), который командой **install** устанавливает Gulp в систему. Ключ **-g** говорит о том, что пакет установится в систему глобально, то-есть в систему, а не в папку проекта. Без ключа **-g** пакет устанавливаются в ту папку, в которой выполняются текущие команды, поэтому будьте внимательны.

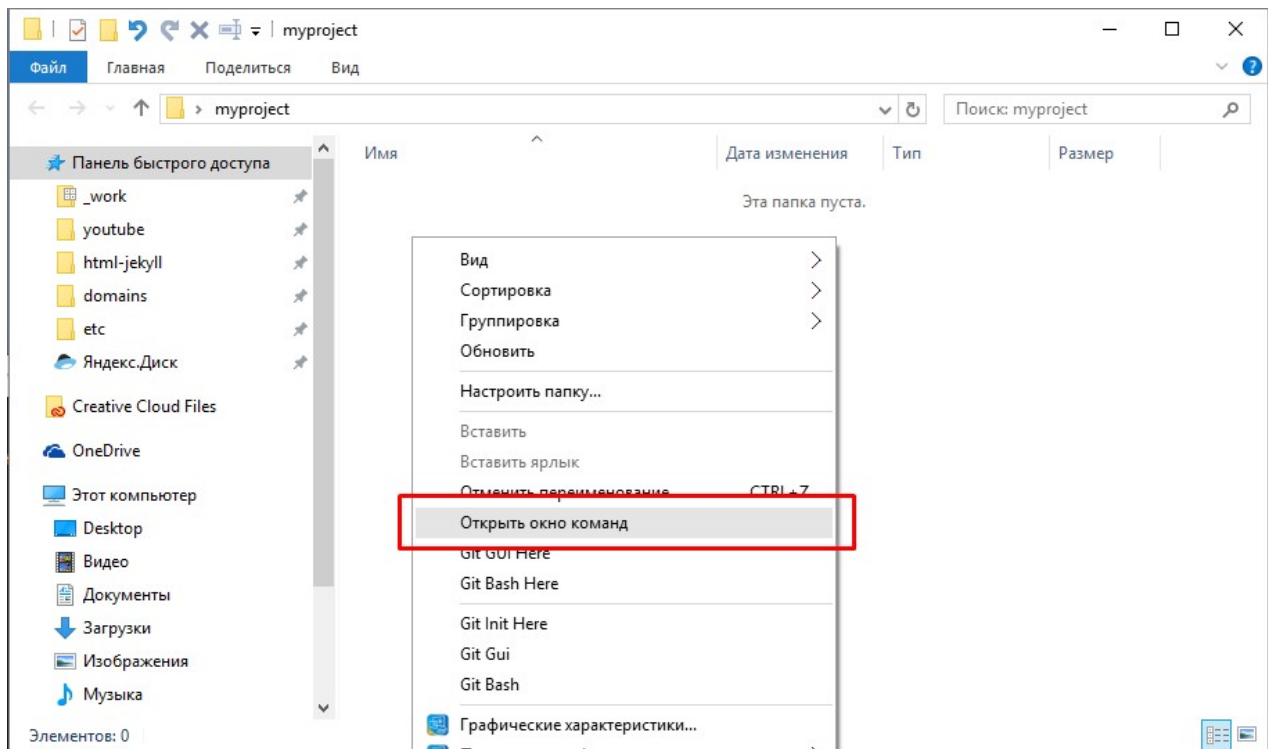
Создание проекта Gulp



Давайте создадим папку проекта для примера, с которой будем работать, пусть это будет, например, папка **myproject**.

Очень важно! Не создавайте русскоязычные папки проектов и следите за тем, чтобы путь до папки проекта не содержал кириллических символов, то-есть не был написан на русском языке. В противном случае, у вас могут возникнуть проблемы при работе различных утилит Gulp. Папка вашего пользователя также не должна быть русскоязычной. Всё только латинскими буквами.

Теперь откроем терминал в папке проекта. Для пользователей Windows достаточно зажать Shift и открыть контекстное меню. В нем появится пункт "Открыть окно команд":

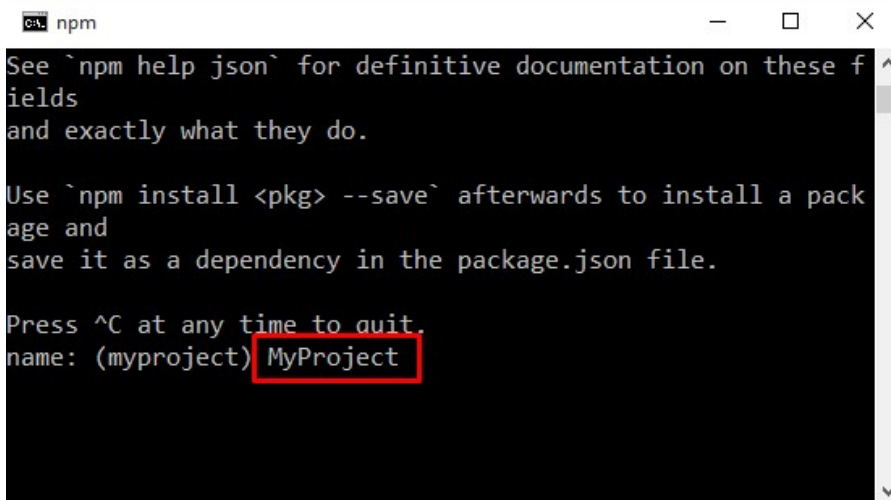


Далее выполним инициализацию проекта в той папке, которую создали:

```
code source
1. npm init
```

Следуя инструкциям, заполним метаинформацию о нашем проекте:

1. Назовем проект "MyProject"

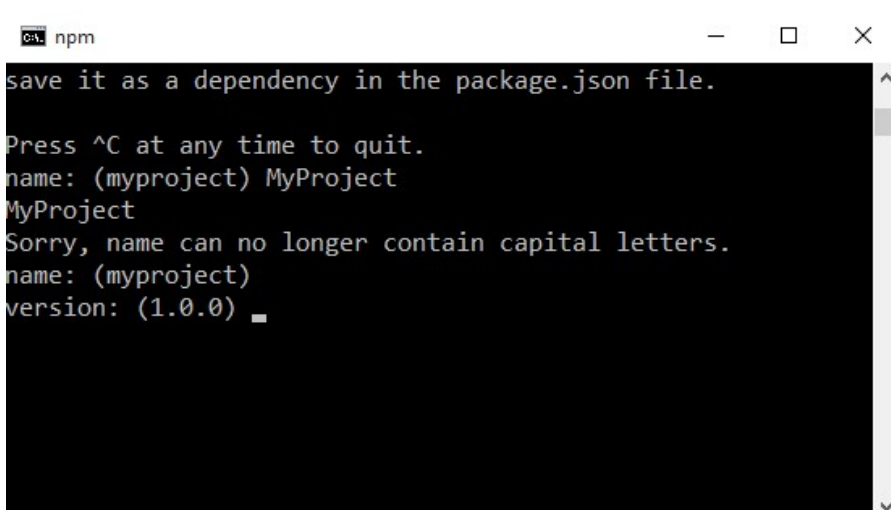


```
npm
See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (myproject) MyProject
```

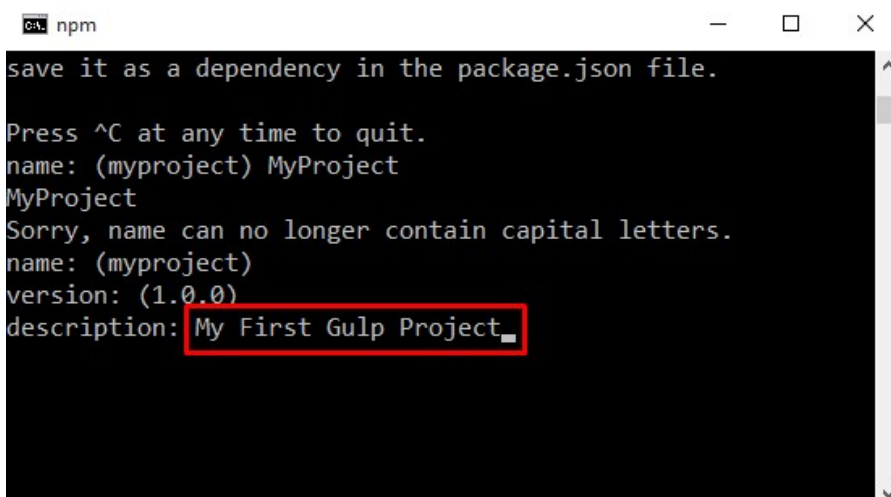
2. Оставим версию текущей - 1.0.0



```
npm
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (myproject) MyProject
MyProject
Sorry, name can no longer contain capital letters.
name: (myproject)
version: (1.0.0)
```

3. Введем краткое описание проекта - My First Gulp Project:



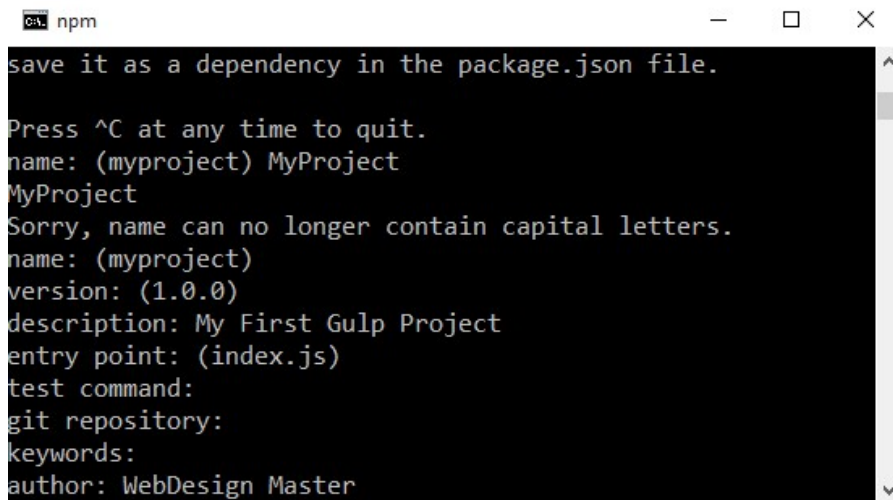
```
npm
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (myproject) MyProject
MyProject
Sorry, name can no longer contain capital letters.
name: (myproject)
version: (1.0.0)
description: My First Gulp Project
```

4. entry point, test command, git repository, keywords оставим по-умолчанию.

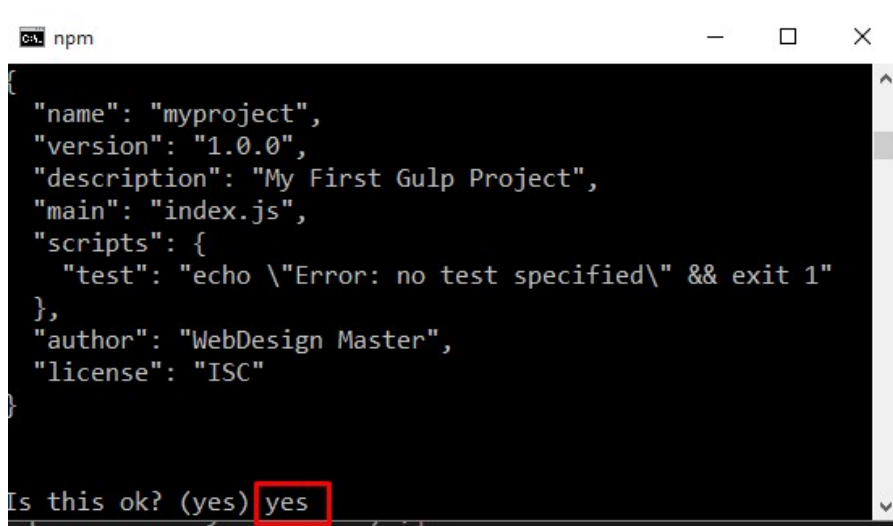
5. Имя автора можно и указать :-)





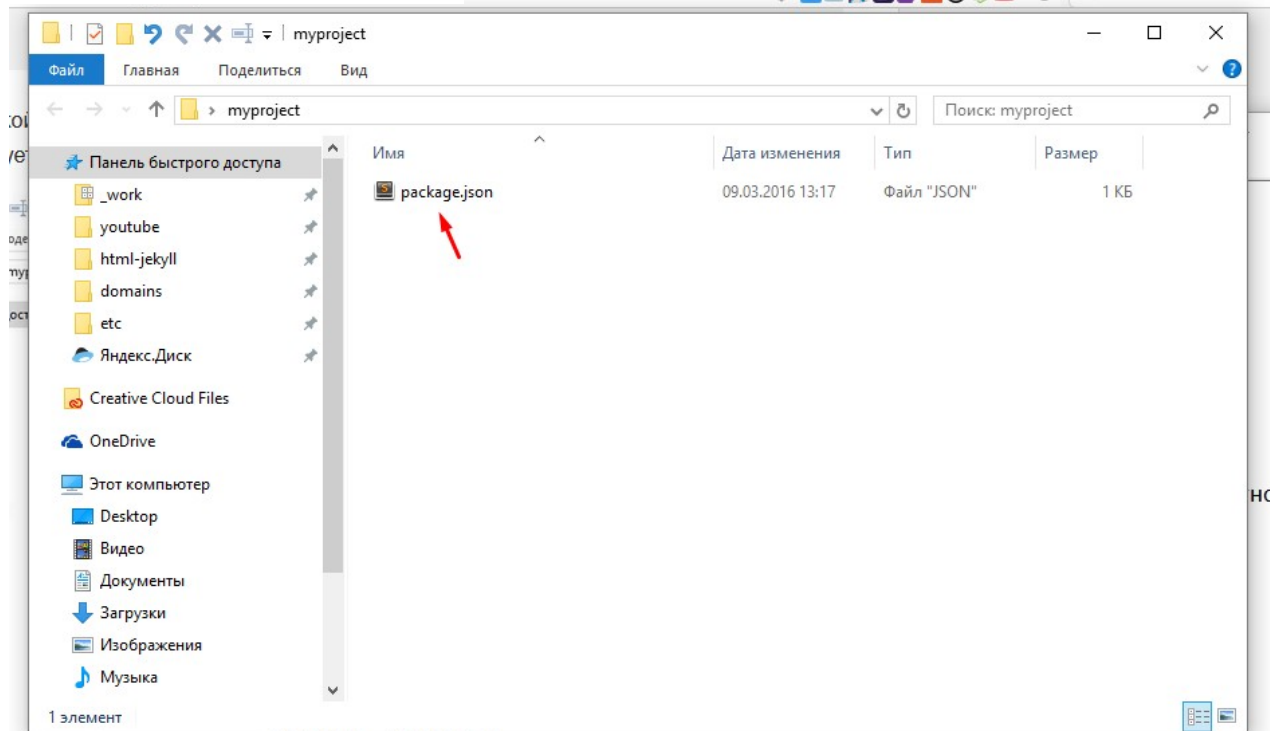
```
save it as a dependency in the package.json file.
Press ^C at any time to quit.
name: (myproject) MyProject
MyProject
Sorry, name can no longer contain capital letters.
name: (myproject)
version: (1.0.0)
description: My First Gulp Project
entry point: (index.js)
test command:
git repository:
keywords:
author: WebDesign Master
```

6. license оставляем по-умолчанию и вводим yes:



```
{
  "name": "myproject",
  "version": "1.0.0",
  "description": "My First Gulp Project",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "WebDesign Master",
  "license": "ISC"
}
Is this ok? (yes) yes
```

В результате такой несложной первоначальной настройки нашего нового Gulp проекта в папке myproject нарисуеться новый файл **package.json**.



Файл `package.json` является файлом манифеста нашего проекта, который описывает помимо той информации, что мы внесли в терминале, еще и информацию об используемых пакетах в нашем проекте.

Например, если мы установим в проект Gulp с ключом `--save-dev`, то пакет и используемая версия автоматически добавится в наш `package.json`. Такой учет позволит быстро разворачивать новый проект с использованием уже имеющегося `package.json` и устанавливать необходимые модули с зависимостями, которые прописаны в `package.json` в новых проектах.

Давайте установим в наш проект Gulp:

```
code source
-----
1. npm i gulp --save-dev
```

Что мы видим из данной строки: `npm` устанавливает пакет `gulp` в текущую папку `myproject` (потому, что нет ключа `-g`, устанавливающий пакет глобально в систему) и сохраняет название пакета с версией в файл `package.json`:

```
1 {
2   "name": "myproject",
3   "version": "1.0.0",
4   "description": "My First Gulp Project",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "WebDesign Master",
10  "license": "ISC",
11  "devDependencies": {
12    "gulp": "^3.9.1"
13  }
14 }
```


Кроме того, у нас появляется папка **node_modules**, которая теперь содержит установленный пакет gulp и необходимые зависимости. В данную папку автоматически будут сваливаться все модули и зависимости, которые мы будем устанавливать в проект. Папок с зависимостями может быть очень много, не смотря на то, что мы установили не так уж и много пакетов. Это связано с тем, что в дополнение к основным пакетам устанавливаются программы, необходимые для корректной работы основного пакета. Ни чего чистить и удалять из папки node_modules не нужно.

Общепринятая структура каталогов в проектах

Работая с различными плагинами, программами и скриптами, будь то jQuery плагин, модуль для CMS, веб-проект или какое-то другое ПО, вы наверняка замечали, что у всех проектов есть общая структура каталогов, например, большинство проектов имеет папку **dist** и **app**. Давайте создадим первоначальную структуру нашего учебного проекта по всем правилам хорошего тона веб-разработки. В результате мы должны создать следующую структуру в нашем проекте myproject (все файлы, которых не было, пока создаем пустыми):

- **myproject/**
 - **app/**
 - **css/**
 - **fonts/**
 - **img/**
 - **js/**
 - **sass/**
 - **index.html**
 - **dist/**
 - **node_modules/**
 - **gulpfile.js**
 - **package.json**

Данная структура встречается довольно часто, практически во всех проектах, но это не аксиома и некоторые проекты могут иметь вообще другую структуру. Для данной статьи мы будем использовать именно такую структуру проекта.

Здесь мы видим папку **app/**, в которой будут размещены все исходные материалы проекта - оригинальные CSS, Sass, js файлы библиотек, оригинальные изображения. В общем - это папка исходников нашего проекта.

Папка **dist/** будет содержать уже готовый продукт, оптимизированный, сжатый, причесанный. Это папка продакшена.

gulpfile.js

Теперь давайте откроем в редакторе кода **gulpfile.js** и напишем в него:

code source

```
1. var gulp = require('gulp');
```



Данной строчкой мы подключаем Gulp к нашему проекту, посредством функции **require**. Данная функция подключает пакеты из папки `node_modules` в наш проект, присваивая их переменной. В данном случае, мы создаем переменную **gulp**.

Далее мы уже можем работать с этой переменной и создавать задачи (инструкции).

```
code source
1. gulp.task('mytask', function() {
2.   console.log('Привет, я task!');
3. });
```

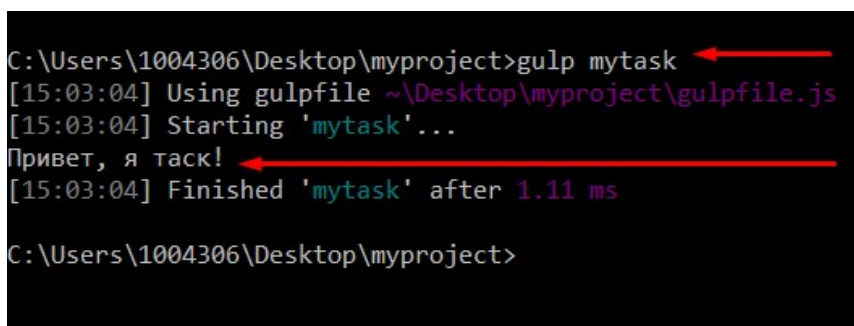
mytask - это название команды, которую вы будете вызывать в нужном вам месте `gulpfile.js`. Кроме того, можно в командной строке выполнить task напрямую, написав:

```
code source
1. gulp mytask
```

gulpfile.js:



Результат выполнения команды **gulp mytask**:



Это, конечно очень простой базовый пример создания taska. Как правило, taskи несколько сложнее и включают некоторые дополнительные команды:

```
code source
1. gulp.task('mytask', function () {
2.   return gulp.src('source-files') // Выборка исходных файлов для обработки плагином
3.     .pipe(plugin()) // Вызов Gulp плагина для обработки файла
4.     .pipe(gulp.dest('folder')) // Вывод результирующего файла в папку назначения (dest - пункт назна
5. });
```



- **gulp.src** - выборка исходных файлов проекта для обработки плагином;
- **.pipe(plugin())** - вызов Gulp плагина для обработки файла;
- **.pipe(gulp.dest('folder'))** - вывод результирующего файла в папку назначения (dest - пункт назначения).

Это база Gulp, теперь можно создавать инструкции. Для начала давайте создадим обработчик, который будет компилировать Sass файлы в CSS (CSS препроцессинг).

Gulp Sass

Давайте установим пакет **gulp-sass** в наш проект с сохранением версии и названия в package.json.

Обратите внимание, что любые Gulp пакеты, для любых задач, легко гуглятся и имеют вполне исчерпывающие инструкции по подключению на своих хоумпейджах и в документации.

code source

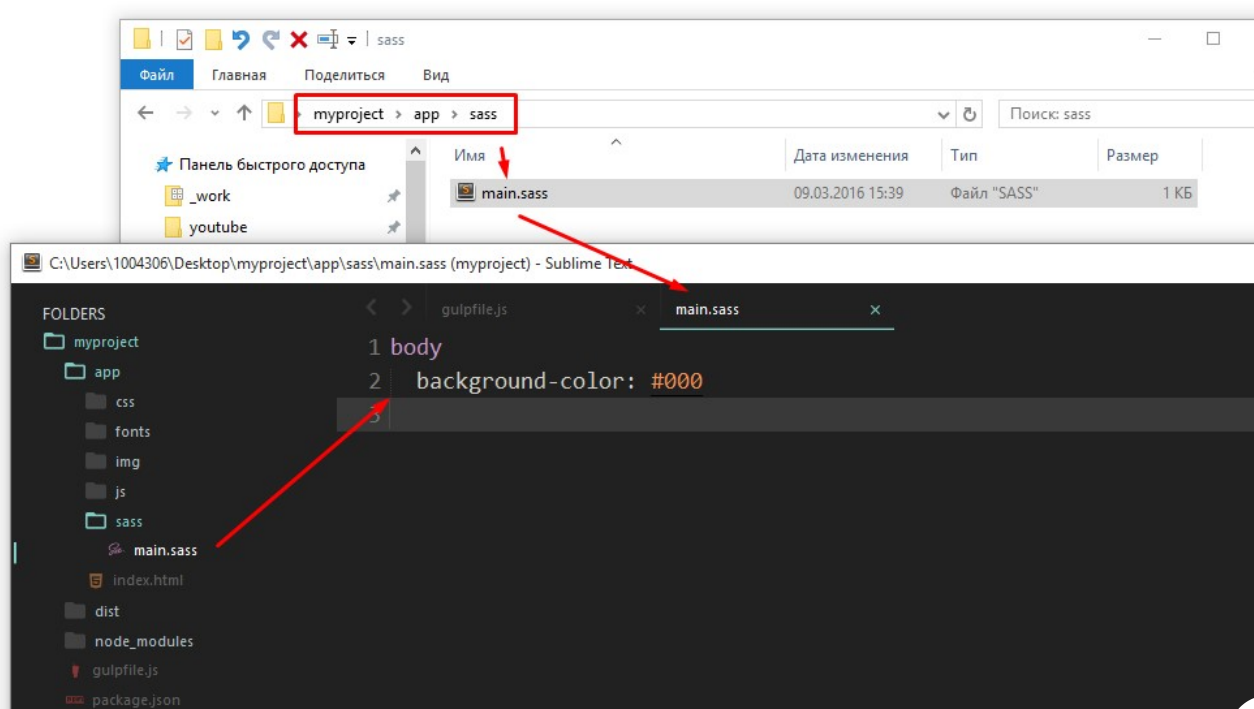
```
1. npm i gulp-sass --save-dev
```

Далее подключим gulp-sass в файле gulpfile.js. Обратите внимание, что переменные для подключения пакетов можно перечислять через запятую:

code source

```
1. var gulp = require('gulp'),
2.   sass = require('gulp-sass'); //Подключаем Sass пакет
```

Давайте создадим в папке app/sass файл **main.sass**, зададим в нем фон body - черный и напомним для него обработчик в gulpfile.js



gulpfile.js:

code source

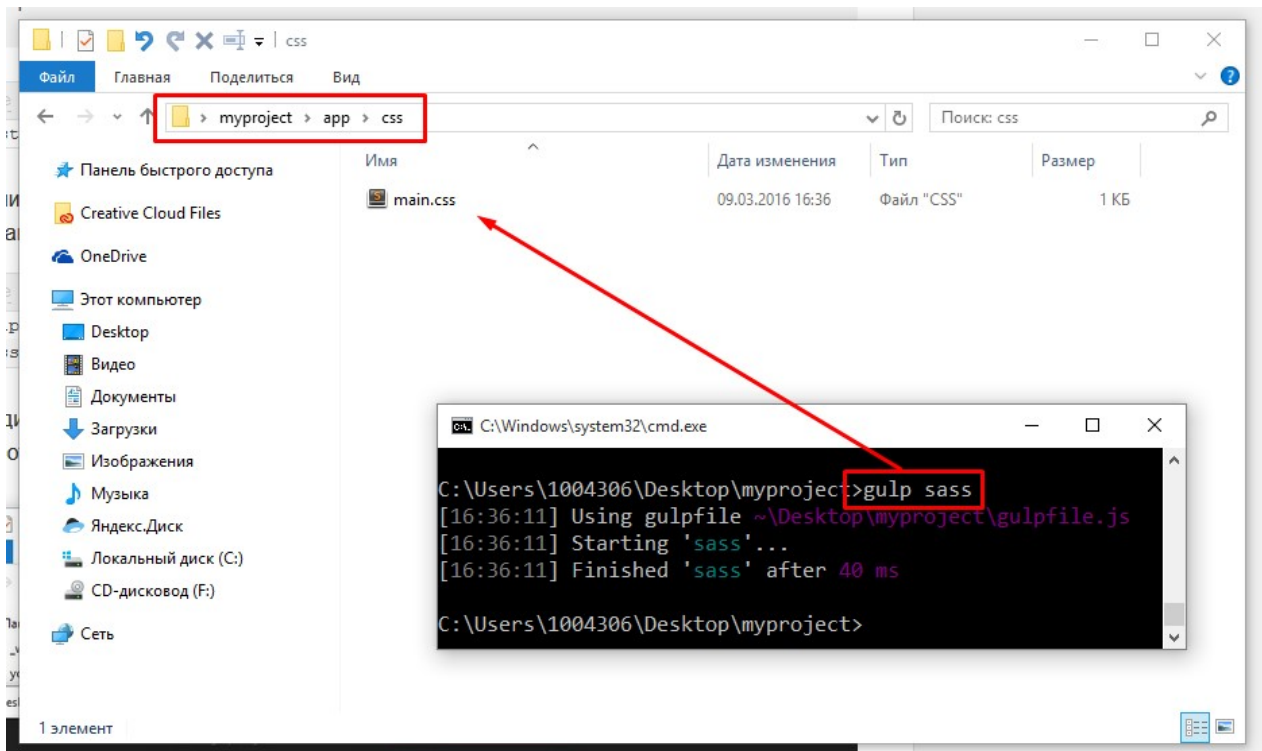
```
1. var gulp = require('gulp'), // Подключаем Gulp
2.   sass = require('gulp-sass'); //Подключаем Sass пакет
3.
4. gulp.task('sass', function(){ // Создаем таск "sass"
5.   return gulp.src('app/sass/main.sass') // Берем источник
6.     .pipe(sass()) // Преобразуем Sass в CSS посредством gulp-sass
7.     .pipe(gulp.dest('app/css')) // Выгружаем результата в папку app/css
8. });
```

После этого, логичным будет выполнить в терминале наш новый таск **sass**:

code source

```
1. gulp sass
```

В результате выполнения данной команды в папке app/css появится файл **main.css**.



От таки чудеса, друзья. Как видим, все просто :-)

Выборка файлов для gulp.src

В принципе, мы рассмотрели все, что необходимо знать о Gulp, теперь будем углубляться в каждую деталь того, что было изложено выше.

Выборка файлов в примере выше довольно простая, мы брали файл напрямую:

gulp.src('app/sass/main.sass'). Но файлы также можно выбирать по шаблону. Шаблон выборки файлов называется **glob** - [https://en.wikipedia.org/wiki/Glob_\(programming\)](https://en.wikipedia.org/wiki/Glob_(programming)). Давайте познакомимся ближе со всеми возможностями выборки файлов для обработки.

Самые распространенные шаблоны выборки

- ***.sass** - выбирает все файлы, имеющие определенное расширение (в данном случае, .sass) в **корневой папке** проекта.
- ****/*.js** - выбирает все файлы с расширением .js во всех папках проекта.



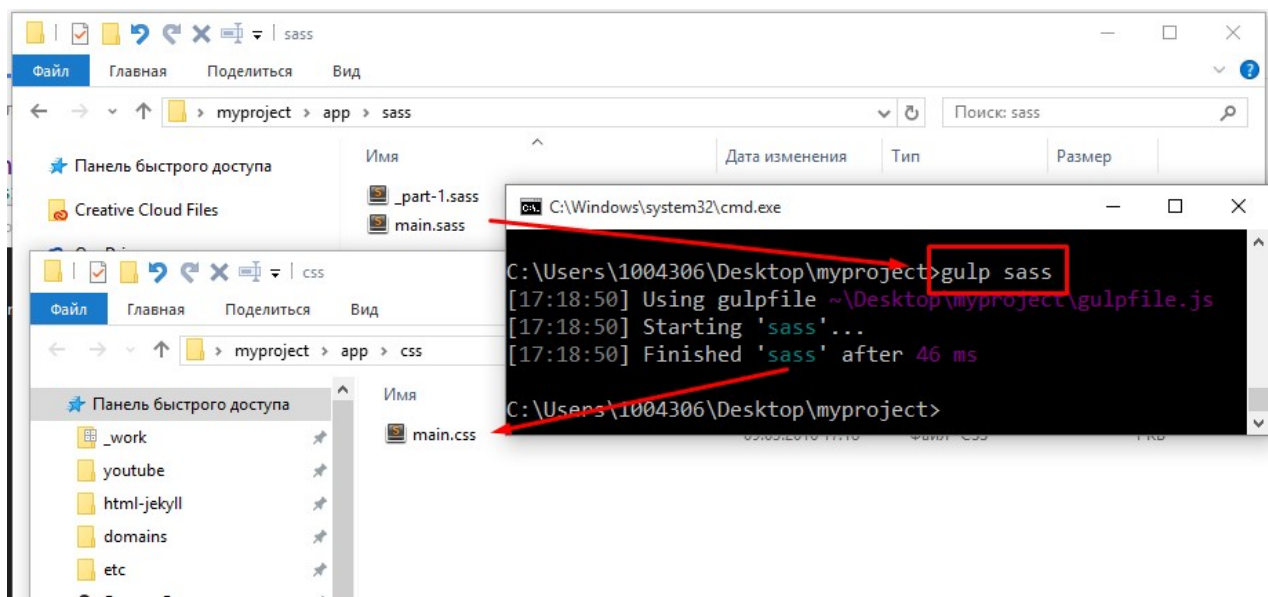
- **!header.sass** - исключает файл из общей выборки
- ***.+(scss|sass)** - задает комплексный шаблон для нескольких типов файлов, разделенных вертикальной чертой. В данном примере в выборку попадут любые sass и scss файлы в корне проекта.

Давайте внесем некоторые изменения в task **sass** и сделаем его более универсальным:

```
code source
1. gulp.task('sass', function(){
2.     return gulp.src('app/sass/**/*.sass') // Берем все sass файлы из папки sass и дочерних, если тако
3.         .pipe(sass())
4.         .pipe(gulp.dest('app/css'))
5. });
```

Дело в том, что брать напрямую один отдельный файл не всегда удобно, так как в папке **sass** могут появиться и другие файлы с расширением sass, которые могут использоваться в проекте.

Обратите внимание, что файлы sass, которые предназначены для импорта в другие файлы, как части одного общего, начинаются с нижнего подчеркивания **_part-1.sass**. Такие файлы не участвуют в компиляции, как отдельные файлы, а добавляются через **@import** в основные файлы.



Наблюдение за изменениями в файлах (Gulp Watch)

Gulp поддерживает метод **watch** для проверки сохраняемых файлов и имеет следующий синтаксис:

```
code source
1. gulp.watch('watch-files', ['task1', 'task2']);
```

Если мы, например, хотим наблюдать за всеми изменениями в файлах sass нашего проекта, то можем использовать следующую конструкцию:

```
code source
1. gulp.watch('app/sass/**/*.sass', ['sass']);
```



Что мы видим: Gulp наблюдает за всеми sass файлами и при сохранении выполняет task sass, который автоматически компилирует их в css файлы.

Также, мы можем создать отдельный task для наблюдения за всеми необходимыми файлами

```
code source
1. gulp.task('watch', function() {
2.   gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за sass файлами
3.   // Наблюдение за другими типами файлов
4. });
```

Если мы запустим в консоли **gulp watch**, то Gulp будет автоматически следить за всеми изменениями в файлах sass при сохранении и компилировать их в css.

Было бы неплохо в дополнение к этой красоте сделать автоматическую перезагрузку страницы при изменениях в файлах. Для этой задачи нам подойдет [Browser Sync](#).

Автоматическое обновление страниц с использованием Browser Sync

Browser Sync - это отличное решение для LiveReload страниц при сохранении файлов. При чем релоад происходит не только в одном браузере, но и во всех браузерах сети, будь это мобильные устройства или другие компьютеры в одной Wi-Fi сети.

Мы уже умеем устанавливать дополнения для Gulp, поэтому давайте установим Browser Sync в наш проект:

```
code source
1. npm i browser-sync --save-dev
```

И, конечно-же, подключим в файле gulpfile.js, как мы это делали ранее с пакетом gulp-sass.

```
code source
1. var gulp      = require('gulp'), // Подключаем Gulp
2.   sass        = require('gulp-sass'), //Подключаем Sass пакет,
3.   browserSync = require('browser-sync'); // Подключаем Browser Sync
```

Создаем task для Browser Sync:

```
code source
1. gulp.task('browser-sync', function() { // Создаем task browser-sync
2.   browserSync({ // Выполняем browser Sync
3.     server: { // Определяем параметры сервера
4.       baseDir: 'app' // Директория для сервера - app
5.     },
6.     notify: false // Отключаем уведомления
7.   });
8. });
```

Отлично! Наш сервер для работы и автоматического релоада готов. Теперь давайте последуем за изменениями в Sass. Если файл Sass обновляется, автоматически инжектируем в HTML измененный CSS файл:

```
code source
1. gulp.task('sass', function(){ // Создаем task Sass
2.   return gulp.src('app/sass/**/*.sass') // Берем источник
3.     .pipe(sass()) // Преобразуем Sass в CSS посредством gulp-sass
4.     .pipe(gulp.dest('app/css')) // Выгружаем результата в папку app/css
5.     .pipe(browserSync.reload({stream: true})) // Обновляем CSS на странице при изменении
```



```
6. });
```

Все, что нам осталось сделать - это запустить task `browser-sync` перед тем, как запустится **gulp watch**. Немного модифицируем task **watch**, добавив выполнение **browser-sync** и **sass** до запуска **watch**:

```
code source
```

```
1. gulp.task('watch', ['browser-sync', 'sass'], function() {
2.     gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за sass файлами
3.     // Наблюдение за другими типами файлов
4. });
```

Обратите внимание, что мы выполняем задачи `['browser-sync', 'sass']` до запуска **watch**, так как их выполнение необходимо нам для корректного отображения изменений на момент запуска сервера.

Расположим task `watch` после всех других задач и в результате получим такой **gulpfile.js**:

```
code source
```

```
1. var gulp      = require('gulp'), // Подключаем Gulp
2.     sass       = require('gulp-sass'), //Подключаем Sass пакет,
3.     browserSync = require('browser-sync'); // Подключаем Browser Sync
4.
5. gulp.task('sass', function(){ // Создаем task Sass
6.     return gulp.src('app/sass/**/*.sass') // Берем источник
7.         .pipe(sass()) // Преобразуем Sass в CSS посредством gulp-sass
8.         .pipe(gulp.dest('app/css')) // Выгружаем результата в папку app/css
9.         .pipe(browserSync.reload({stream: true})) // Обновляем CSS на странице при изменении
10. });
11.
12. gulp.task('browser-sync', function() { // Создаем task browser-sync
13.     browserSync({ // Выполняем browserSync
14.         server: { // Определяем параметры сервера
15.             baseDir: 'app' // Директория для сервера - app
16.         },
17.         notify: false // Отключаем уведомления
18.     });
19. });
20.
21. gulp.task('watch', ['browser-sync', 'sass'], function() {
22.     gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за sass файлами
23.     // Наблюдение за другими типами файлов
24. });
```

Для того, чтобы следить за изменениями в браузере, сделаем соответствующую разметку в файле `index.html` директории `app` с подключением файла стилей **main.css**:

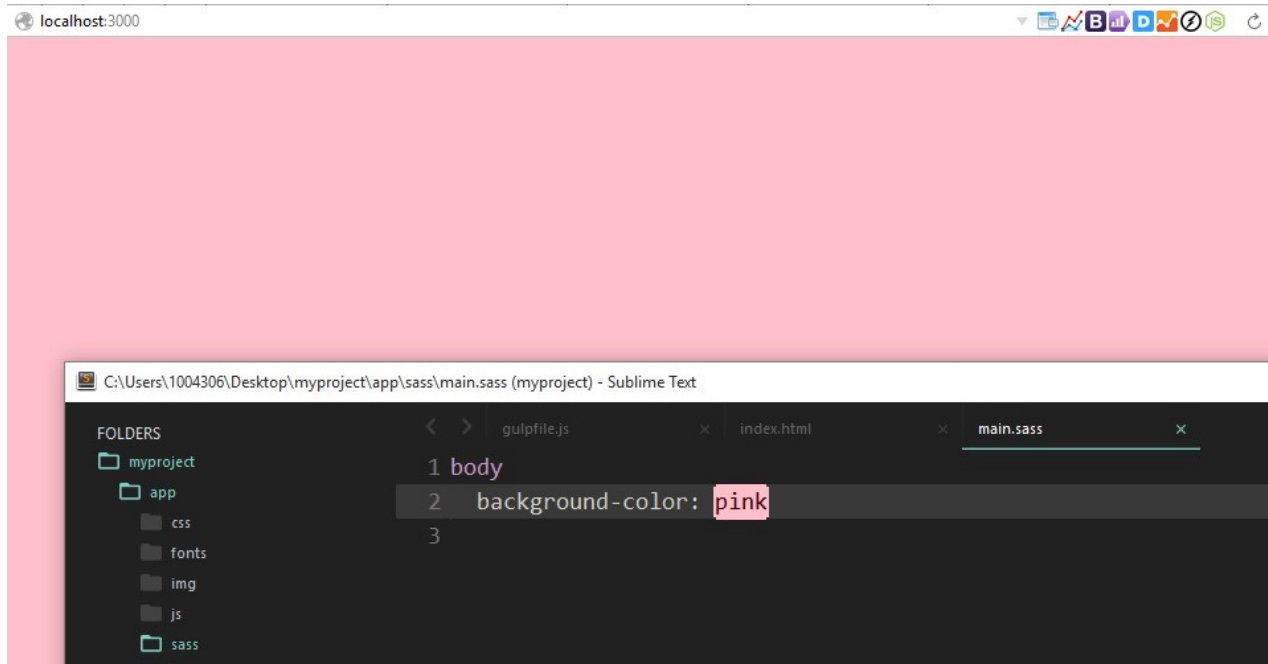
```
code source
```

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>Document</title>
6.     <link rel="stylesheet" href="css/main.css"> <!-- Подключаем CSS -->
7. </head>
8. <body>
9.
10. </body>
```

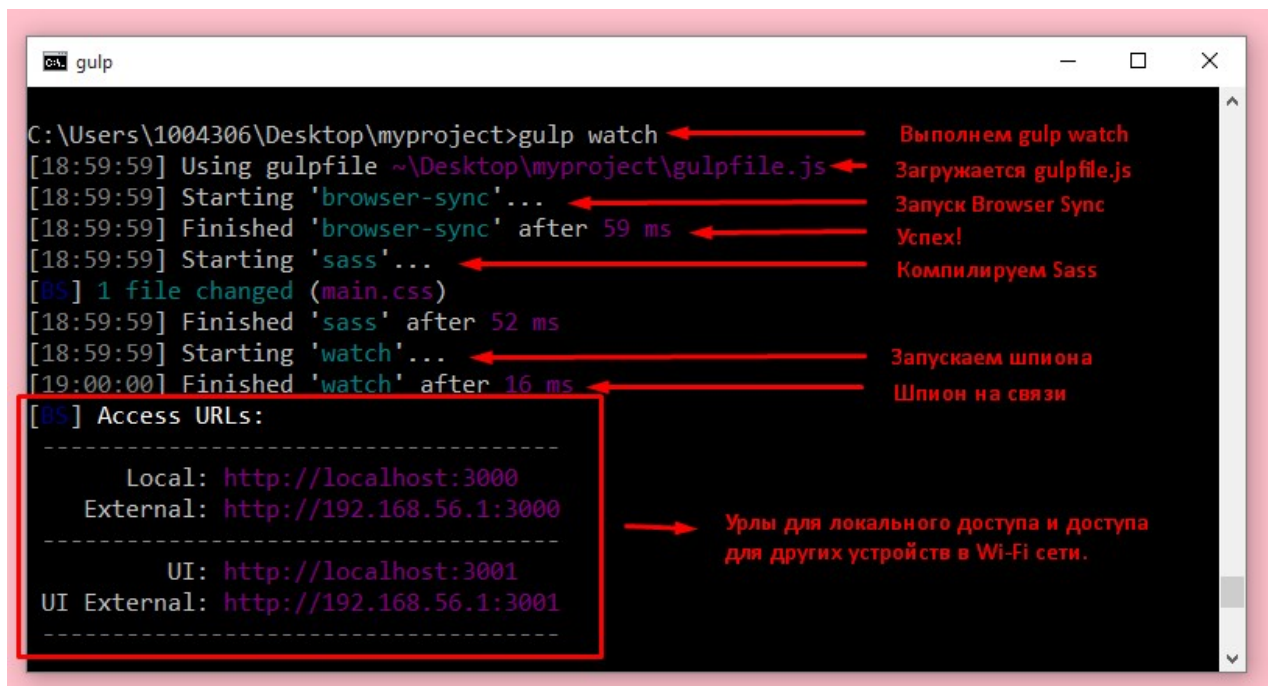


11. </html>

Результат заворачивает:



Давайте разберемся, что у нас происходит в консоли:



После того, как мы порадуемся результату, встает весьма ожидаемый вопрос - а как, собственно, обновлять страницу при сохранении HTML и JS?

И эта задача нам по плечу:

code source

```
1. gulp.task('watch', ['browser-sync', 'sass'], function() {
2.   gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за sass файлами в папке sass
3.   gulp.watch('app/*.html', browserSync.reload); // Наблюдение за HTML файлами в корне проекта
4.   gulp.watch('app/js/**/*.js', browserSync.reload); // Наблюдение за JS файлами в папке js
5. });
```



Здесь мы используем функцию `browserSync.reload`, которую нам любезно предоставил пакет `Browser Sync`. Обратите внимание на выборку файлов для слежения.

В принципе, мы уже имеем довольно продвинутое рабочее окружение. Но двигаемся дальше, это не все, на что способен Gulp.

Оптимизация JavaScript

Давайте рассмотрим, как можно оптимизировать JS файлы проекта. Чаще всего, в оптимизации нуждаются библиотеки и сторонние jQuery и JavaScript плагины. Давайте создадим в папке **app** паку **libs**, которая будет содержать необходимые проекту библиотеки. Все библиотеки будем размещать в отдельных папках. Для установки новых библиотек я советую использовать [Bower](#).

Установим Bower:

```
code source
1. npm i -g bower
```

Обратите внимание, что для работы Bower необходим установленный [Git](#).

Теперь в папке проекта создадим файл **.bowerrc**, в который напомним:

```
code source
1. {
2.   "directory" : "app/libs/"
3. }
```

Если вы пользователь ОС Windows, у вас не получится создать файл, начинающийся с точки. Воспользуйтесь FileZilla или каким-нибудь файловым менеджером для этой задачи.

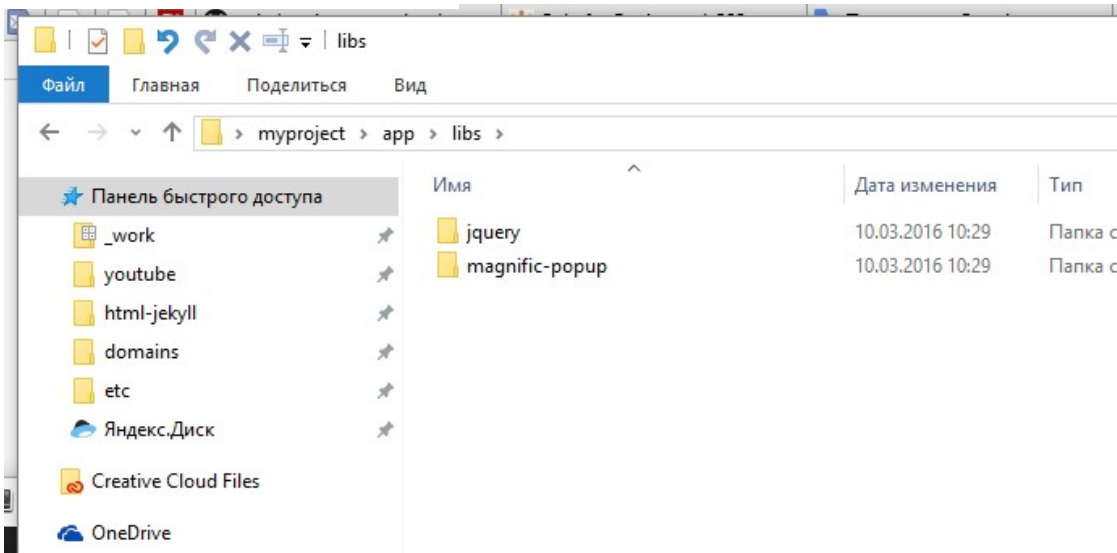
Данной настройкой мы указываем путь по умолчанию для установки плагинов с помощью **Bower**.

Установим jQuery и Magnific Popup, для примера:

```
code source
1. bower i jquery magnific-popup
```

Круть:





Обратите внимание, что все (ну, или почти все) плагины имеют папку `dist`, об этом мы говорили ранее. В этой папке располагаются готовые файлы продакшена, которые мы и будем использовать в нашем проекте.

Давайте создадим task **scripts**, который будет собирать все JS файлы библиотек в один и минифицировать файл. Для этого установим 2 пакета: **gulp-concat** и **gulp-uglifyjs**.

code source

```
1. npm i --save-dev gulp-concat gulp-uglifyjs
```

Подключим новые библиотеки в `gulpfile.js`:

code source

```
1. var gulp      = require('gulp'), // Подключаем Gulp
2.    sass       = require('gulp-sass'), //Подключаем Sass пакет,
3.    browserSync = require('browser-sync'), // Подключаем Browser Sync
4.    concat     = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.    uglify     = require('gulp-uglifyjs'); // Подключаем gulp-uglifyjs (для сжатия JS)
```

Создаем задачу для сборки и сжатия всех библиотек (перед watch):

code source

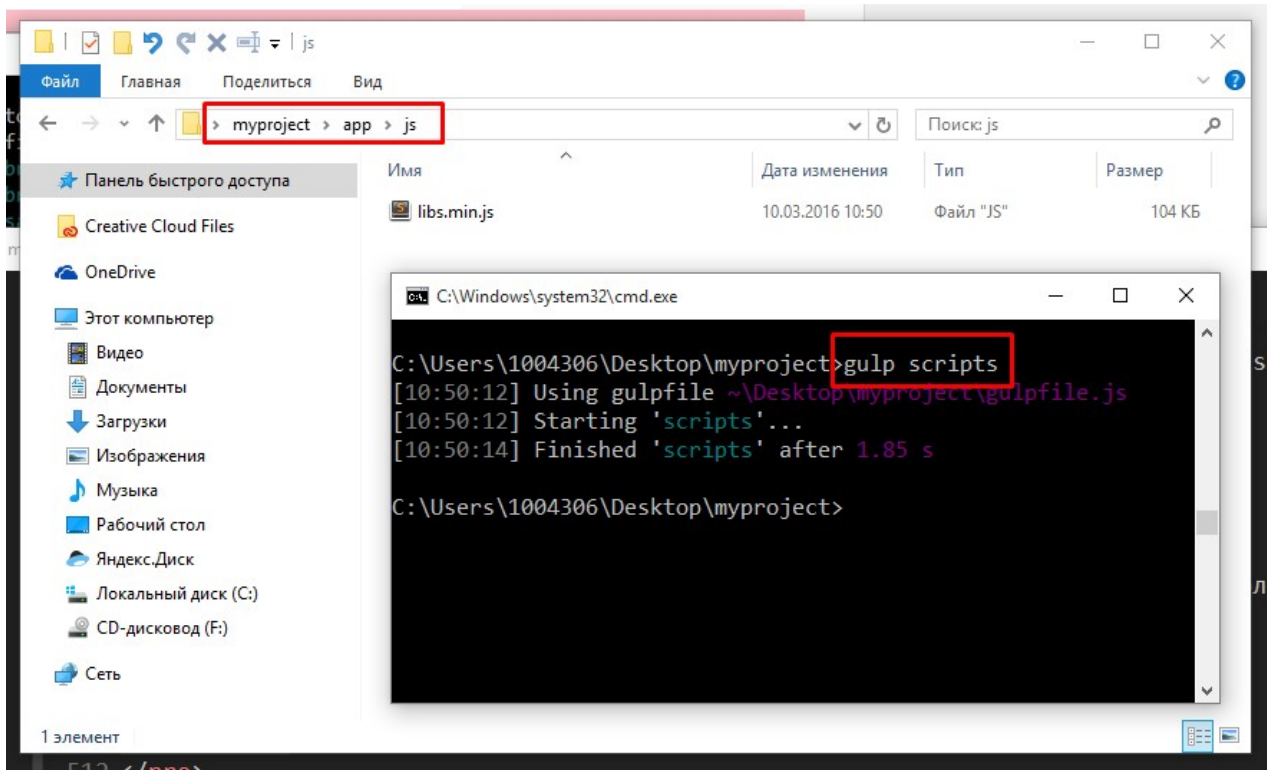
```
1. gulp.task('scripts', function() {
2.   return gulp.src([ // Берем все необходимые библиотеки
3.     'app/libs/jquery/dist/jquery.min.js', // Берем jQuery
4.     'app/libs/magnific-popup/dist/jquery.magnific-popup.min.js' // Берем Magnific Popup
5.   ])
6.   .pipe(concat('libs.min.js')) // Собираем их в кучу в новом файле libs.min.js
7.   .pipe(uglify()) // Сжимаем JS файл
8.   .pipe(gulp.dest('app/js')); // Выгружаем в папку app/js
9. });
```

Давайте проверим, как работает наш новый task **scripts**, выполнив в терминале:

code source

```
1. gulp scripts
```





Выполнение задачи **scripts** можно запустить перед выполнением **watch**:

```
gulp.task('watch', ['browser-sync', 'sass', 'scripts'], function() {
  gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за
  gulp.watch('app/*.html', browserSync.reload); // Наблюдение за
  gulp.watch('app/js/**/*.js', browserSync.reload); // Наблюдение за
```

Далее можно подключить к проекту все необходимые CSS файлы библиотек. В нашем случае, только одна библиотека нуждается в подключении - это Magnific Popup. Сделаем это через **@import** в Sass файле **sass/libs.sass**:

```
code source
1. @import "app/libs/magnific-popup/dist/magnific-popup.css" // Импортируем библиотеку Magnific Popup
```

Внимание! В новых версиях **gulp-sass** для импорта CSS файлов в Sass необходимо указывать расширение **.css**

На выходе, в папке **app/css** мы получаем дополнительно к **main.css** файл **libs.css**, который содержит стили всех библиотек. Файл **main.css** нет особого смысла минифицировать, так как он содержит кастомные (пользовательские) стили. А вот файл **libs.css** мы с удовольствием минифицируем.

Для минификации CSS установим пакеты **gulp-cssnano** и **gulp-rename**:

```
code source
1. npm i gulp-cssnano gulp-rename --save-dev
```

И подключим их в нашем **gulpfile.js**:

```
code source
1. var gulp = require('gulp'), // Подключаем Gulp
2. sass = require('gulp-sass'), // Подключаем Sass пакет,
```

```

3.   browserSync = require('browser-sync'), // Подключаем Browser Sync
4.   concat      = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.   uglify       = require('gulp-uglifyjs'), // Подключаем gulp-uglifyjs (для сжатия JS)
6.   cssnano      = require('gulp-cssnano'), // Подключаем пакет для минификации CSS
7.   rename       = require('gulp-rename'); // Подключаем библиотеку для переименования файлов

```

И создадим соответствующий task **css-libs**. Сразу добавим данный task в watch для того, чтобы библиотеки собирались в процессе запуска проекта. Task sass лучше вызвать до запуска css-libs, чтобы нам было что минифицировать:

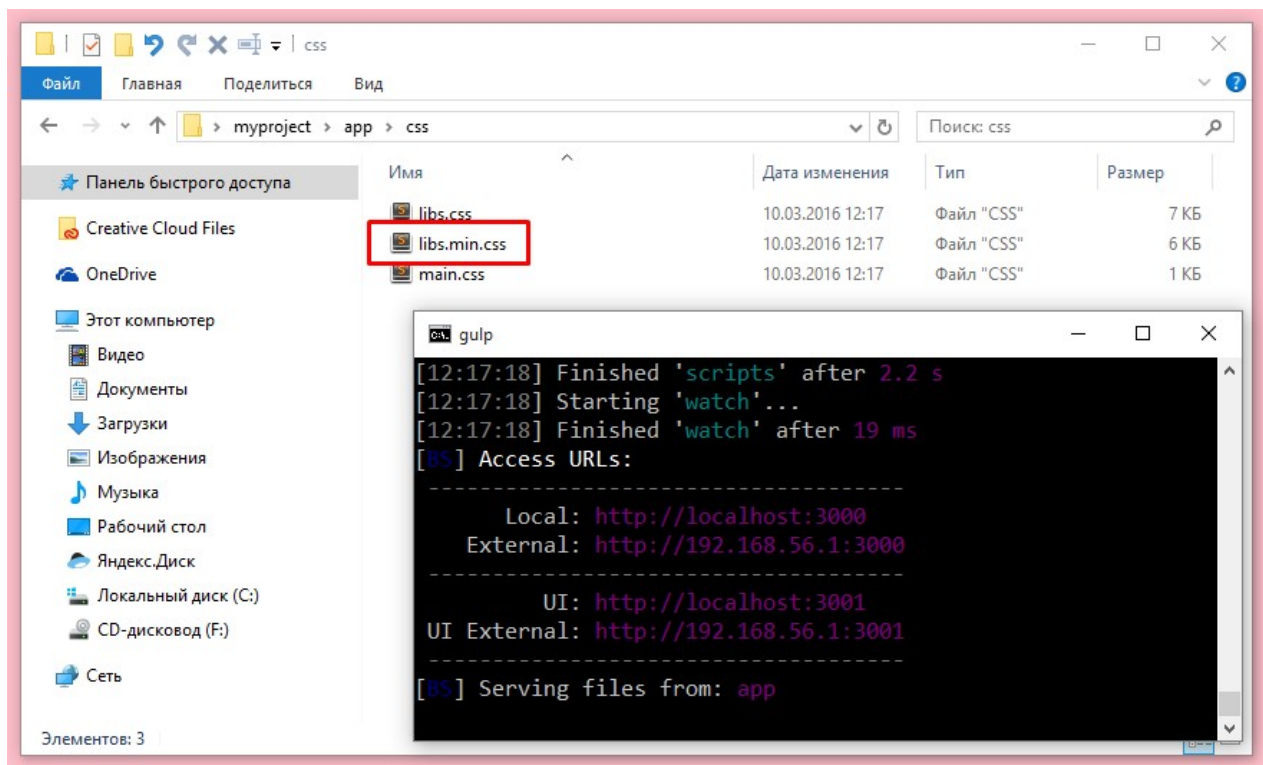
code source

```

1.   gulp.task('css-libs', ['sass'], function() {
2.       return gulp.src('app/css/libs.css') // Выбираем файл для минификации
3.           .pipe(cssnano()) // Сжимаем
4.           .pipe(rename({suffix: '.min'})) // Добавляем суффикс .min
5.           .pipe(gulp.dest('app/css')); // Выгружаем в папку app/css
6.   });
7.
8.   gulp.task('watch', ['browser-sync', 'css-libs', 'scripts'], function() {
9.       gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за sass файлами в папке sass
10.      gulp.watch('app/*.html', browserSync.reload); // Наблюдение за HTML файлами в корне проекта
11.      gulp.watch('app/js/**/*.js', browserSync.reload); // Наблюдение за JS файлами в папке js
12.   });

```

Ура:



Подготовка к продакшену

Для продакшена (сборки в папку dist) мы создадим отдельный task build в конце gulpfile.js. В данной инструкции мы осуществим сборку Sass, JS и выгрузку того, что у нас готово в папку dist.

code source

```

1.   gulp.task('build', ['sass', 'scripts'], function() {
2.
3.       var buildCss = gulp.src([ // Переносим CSS стили в продакшен
4.           'app/css/main.css',

```



```

5.     'app/css/libs.min.css'
6.   ])
7.   .pipe(gulp.dest('dist/css'))
8.
9.   var buildFonts = gulp.src('app/fonts/**/*') // Переносим шрифты в продакшен
10.  .pipe(gulp.dest('dist/fonts'))
11.
12.  var buildJs = gulp.src('app/js/**/*') // Переносим скрипты в продакшен
13.  .pipe(gulp.dest('dist/js'))
14.
15.  var buildHtml = gulp.src('app/*.html') // Переносим HTML в продакшен
16.  .pipe(gulp.dest('dist'));
17.
18. });

```

Здесь, присваивая переменным какие-либо действия, мы их выполняем. Таким образом можно выполнять мультизадачные задачи. Можно и не присваивать, но мы сделаем так, ибо красивше.

Все прекрасно, но всегда есть одно "Но". Перед тем, как собирать проект нам желательно бы очистить папку dist, чтобы не оставалось лишних потрохов от предыдущих итераций с нашим проектом.

Установим и подключим пакет del:

code source

```
1. npm i del --save-dev
```

code source

```

1. var gulp      = require('gulp'), // Подключаем Gulp
2.     sass      = require('gulp-sass'), //Подключаем Sass пакет,
3.     browserSync = require('browser-sync'), // Подключаем Browser Sync
4.     concat     = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.     uglify     = require('gulp-uglifyjs'), // Подключаем gulp-uglifyjs (для сжатия JS)
6.     cssnano    = require('gulp-cssnano'), // Подключаем пакет для минификации CSS
7.     rename     = require('gulp-rename'), // Подключаем библиотеку для переименования файлов
8.     del        = require('del'); // Подключаем библиотеку для удаления файлов и папок

```

Создаем задачу очистки **clean** и добавляем его выполнение перед выполнением build:

code source

```

1. gulp.task('clean', function() {
2.   return del.sync('dist'); // Удаляем папку dist перед сборкой
3. });
4.
5. gulp.task('build', ['clean', 'sass', 'scripts'], function() {
6.
7.   var buildCss = gulp.src([ // Переносим библиотеки в продакшен
8.     'app/css/main.css',
9.     'app/css/libs.min.css'
10.   ])
11.   .pipe(gulp.dest('dist/css'))
12.
13.   var buildFonts = gulp.src('app/fonts/**/*') // Переносим шрифты в продакшен
14.   .pipe(gulp.dest('dist/fonts'))
15.
16.   var buildJs = gulp.src('app/js/**/*') // Переносим скрипты в продакшен
17.   .pipe(gulp.dest('dist/js'))
18.
19.   var buildHtml = gulp.src('app/*.html') // Переносим HTML в продакшен
20.   .pipe(gulp.dest('dist'));
21.

```

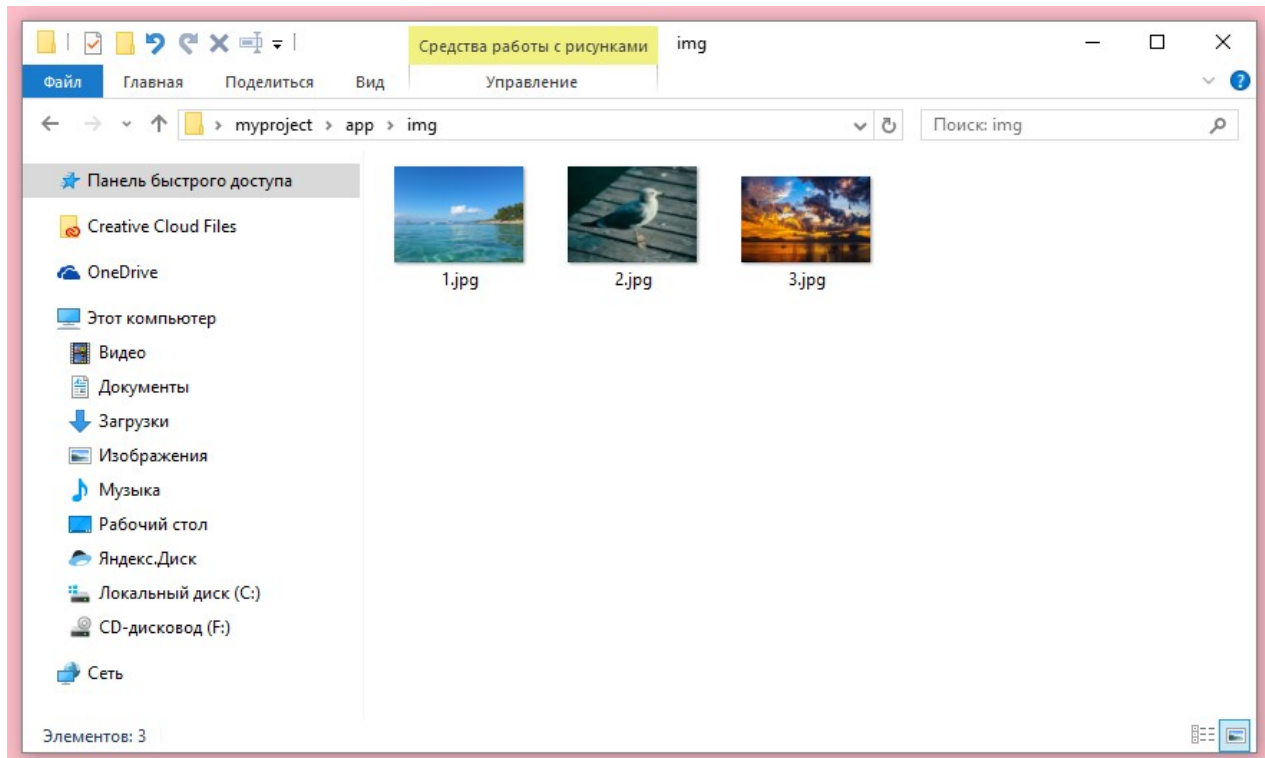


```
22. });
```

Оптимизация изображений

Как вы могли заметить, в нашем проекте на продакшене не хватает изображений. Давайте исправим это недоразумение и добавим обработку изображений в наш проект.

В папке `app/img` есть 3 изображения, которые нам необходимо перенести в папку продакшена, оптимизируя.



Для оптимизации изображений установим 2 пакета (**gulp-imagemin**, **imagemin-pngquant**) и подключим их:

code source

```
1. npm i gulp-imagemin imagemin-pngquant --save-dev
```

code source

```
1. var gulp      = require('gulp'), // Подключаем Gulp
2.     sass       = require('gulp-sass'), //Подключаем Sass пакет,
3.     browserSync = require('browser-sync'), // Подключаем Browser Sync
4.     concat      = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.     uglify       = require('gulp-uglifyjs'), // Подключаем gulp-uglifyjs (для сжатия JS)
6.     cssnano      = require('gulp-cssnano'), // Подключаем пакет для минификации CSS
7.     rename       = require('gulp-rename'), // Подключаем библиотеку для переименования файлов
8.     del          = require('del'), // Подключаем библиотеку для удаления файлов и папок
9.     imagemin     = require('gulp-imagemin'), // Подключаем библиотеку для работы с изображениями
10.    pngquant     = require('imagemin-pngquant'); // Подключаем библиотеку для работы с png
```

Далее создадим таск **img** для сжатия изображений на продакшен и вызовем его после очистки:

code source

```
1. gulp.task('img', function() {
2.     return gulp.src('app/img/**/*') // Берем все изображения из app
3.         .pipe(imagemin({ // Сжимаем их с наилучшими настройками
4.             interlaced: true,
5.             progressive: true,
```




```

6.         svgoPlugins: [{removeViewBox: false}],
7.         use: [pngquant()]
8.     )))
9.     .pipe(gulp.dest('dist/img')); // Выгружаем на продакшен
10. });
11.
12. gulp.task('build', ['clean', 'img', 'sass', 'scripts'], function() {
13.
14.     var buildCss = gulp.src([ // Переносим библиотеки в продакшен
15.         'app/css/main.css',
16.         'app/css/libs.min.css'
17.     ])
18.     .pipe(gulp.dest('dist/css'))
19.
20.     var buildFonts = gulp.src('app/fonts/**/*') // Переносим шрифты в продакшен
21.     .pipe(gulp.dest('dist/fonts'))
22.
23.     var buildJs = gulp.src('app/js/**/*') // Переносим скрипты в продакшен
24.     .pipe(gulp.dest('dist/js'))
25.
26.     var buildHtml = gulp.src('app/*.html') // Переносим HTML в продакшен
27.     .pipe(gulp.dest('dist'));
28.
29. });

```

Все прекрасно. До тех пор, пока количество изображений в проекте не превышает 3 шт. Большое количество картинок будет обрабатываться значительно дольше, поэтому к обработке изображений было бы неплохо добавить кеш, чтобы картинки кешировались, экономя наше время.

Установи и подключим **gulp-cache**:

code [source](#)

```
1. npm i gulp-cache --save-dev
```

code [source](#)

```

1. var gulp      = require('gulp'), // Подключаем Gulp
2.     sass      = require('gulp-sass'), //Подключаем Sass пакет,
3.     browserSync = require('browser-sync'), // Подключаем Browser Sync
4.     concat     = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.     uglify     = require('gulp-uglifyjs'), // Подключаем gulp-uglifyjs (для сжатия JS)
6.     cssnano    = require('gulp-cssnano'), // Подключаем пакет для минификации CSS
7.     rename     = require('gulp-rename'), // Подключаем библиотеку для переименования файлов
8.     del        = require('del'), // Подключаем библиотеку для удаления файлов и папок
9.     imagemin   = require('gulp-imagemin'), // Подключаем библиотеку для работы с изображениями
10.    pngquant    = require('imagemin-pngquant'), // Подключаем библиотеку для работы с png
11.    cache      = require('gulp-cache'); // Подключаем библиотеку кеширования

```

Модифицируем task **img**:

code [source](#)

```

1. gulp.task('img', function() {
2.     return gulp.src('app/img/**/*') // Берем все изображения из app
3.     .pipe(cache(imagemin({ // Сжимаем их с наилучшими настройками с учетом кеширования
4.         interlaced: true,
5.         progressive: true,
6.         svgoPlugins: [{removeViewBox: false}],
7.         use: [pngquant()]
8.     })))
9.     .pipe(gulp.dest('dist/img')); // Выгружаем на продакшен
10. });

```



Автоматическое создание префиксов CSS с помощью Gulp

Вендорные префиксы необходимы для обеспечения максимальной совместимости со всеми современными браузерами. Было бы логично сделать автоматическое добавление префиксов, чтобы написав в CSS или Sass:

```
code source
1. display: flex
```

Мы получили на выходе:

```
code source
1. display: -webkit-flex;
2. display: -moz-flex;
3. display: -ms-flex;
4. display: -o-flex;
5. display: flex;
```

Установим пакет **gulp-autoprefixer** и подключим его в gulpfile.js:

```
code source
1. npm i --save-dev gulp-autoprefixer
```

```
code source
1. var gulp      = require('gulp'), // Подключаем Gulp
2.    sass       = require('gulp-sass'), //Подключаем Sass пакет,
3.    browserSync = require('browser-sync'), // Подключаем Browser Sync
4.    concat     = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.    uglify     = require('gulp-uglifyjs'), // Подключаем gulp-uglifyjs (для сжатия JS)
6.    cssnano    = require('gulp-cssnano'), // Подключаем пакет для минификации CSS
7.    rename     = require('gulp-rename'), // Подключаем библиотеку для переименования файлов
8.    del        = require('del'), // Подключаем библиотеку для удаления файлов и папок
9.    imagemin   = require('gulp-imagemin'), // Подключаем библиотеку для работы с изображениями
10.    pngquant  = require('imagemin-pngquant'), // Подключаем библиотеку для работы с png
11.    cache     = require('gulp-cache'), // Подключаем библиотеку кеширования
12.    autoprefixer = require('gulp-autoprefixer');// Подключаем библиотеку для автоматического добавлен
```

И модифицируем наш task **sass**:

```
code source
1. gulp.task('sass', function(){ // Создаем task Sass
2.     return gulp.src('app/sass/**/*.sass') // Берем источник
3.         .pipe(sass()) // Преобразуем Sass в CSS посредством gulp-sass
4.         .pipe(autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], { cascade: true }))) // Созда
5.         .pipe(gulp.dest('app/css')) // Выгружаем результата в папку app/css
6.         .pipe(browserSync.reload({stream: true})) // Обновляем CSS на странице при изменении
7.     });
```

Дефолтный task Gulp

Итак, мы имеем 2 главных taska - **gulp watch** - для работы над проектом в режиме "онлайн" и **gulp build** - для сборки проекта на продакшен без лишних файлов, папок и со сжатыми картинками. Так как чаще всего нам нужен будет task **watch**, можно повесить его на дефолтный task, чтобы не писать в консоли постоянно **gulp watch**, а писать просто **gulp**.

```
code source
1. gulp.task('default', ['watch']);
```



Также, необходимо создать автономный таск для очистки кеша Gulp, чтобы его можно было вызывать простой командой **gulp clear**:

code source

```
1. gulp.task('clear', function () {
2.     return cache.clearAll();
3. })
```

Если у вас возникнут проблемы с изображениями или другими кешируемыми файлами, просто почистите кеш.

В результате, у нас должен получиться такой **gulpfile.js**:

code source

```
1. var gulp      = require('gulp'), // Подключаем Gulp
2.     sass      = require('gulp-sass'), //Подключаем Sass пакет,
3.     browserSync = require('browser-sync'), // Подключаем Browser Sync
4.     concat     = require('gulp-concat'), // Подключаем gulp-concat (для конкатенации файлов)
5.     uglify     = require('gulp-uglifyjs'), // Подключаем gulp-uglifyjs (для сжатия JS)
6.     cssnano    = require('gulp-cssnano'), // Подключаем пакет для минификации CSS
7.     rename     = require('gulp-rename'), // Подключаем библиотеку для переименования файлов
8.     del        = require('del'), // Подключаем библиотеку для удаления файлов и папок
9.     imagemin   = require('gulp-imagemin'), // Подключаем библиотеку для работы с изображениями
10.    pngquant    = require('imagemin-pngquant'), // Подключаем библиотеку для работы с png
11.    cache      = require('gulp-cache'), // Подключаем библиотеку кеширования
12.    autoprefixer = require('gulp-autoprefixer');// Подключаем библиотеку для автоматического добавлен
13.
14. gulp.task('sass', function(){ // Создаем таск Sass
15.     return gulp.src('app/sass/**/*.sass') // Берем источник
16.         .pipe(sass()) // Преобразуем Sass в CSS посредством gulp-sass
17.         .pipe(autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], { cascade: true }))) // Созда
18.         .pipe(gulp.dest('app/css')) // Выгружаем результата в папку app/css
19.         .pipe(browserSync.reload({stream: true})) // Обновляем CSS на странице при изменении
20. });
21.
22. gulp.task('browser-sync', function() { // Создаем таск browser-sync
23.     browserSync({ // Выполняем browserSync
24.         server: { // Определяем параметры сервера
25.             baseDir: 'app' // Директория для сервера - app
26.         },
27.         notify: false // Отключаем уведомления
28.     });
29. });
30.
31. gulp.task('scripts', function() {
32.     return gulp.src([ // Берем все необходимые библиотеки
33.         'app/libs/jquery/dist/jquery.min.js', // Берем jQuery
34.         'app/libs/magnific-popup/dist/jquery.magnific-popup.min.js' // Берем Magnific Popup
35.     ])
36.         .pipe(concat('libs.min.js')) // Собираем их в кучу в новом файле libs.min.js
37.         .pipe(uglify()) // Сжимаем JS файл
38.         .pipe(gulp.dest('app/js')); // Выгружаем в папку app/js
39. });
40.
41. gulp.task('css-libs', ['sass'], function() {
42.     return gulp.src('app/css/libs.css') // Выбираем файл для минификации
43.         .pipe(cssnano()) // Сжимаем
44.         .pipe(rename({suffix: '.min'})) // Добавляем суффикс .min
45.         .pipe(gulp.dest('app/css')); // Выгружаем в папку app/css
46. });
47.
48. gulp.task('watch', ['browser-sync', 'css-libs', 'scripts'], function() {
```



```
49.   gulp.watch('app/sass/**/*.sass', ['sass']); // Наблюдение за sass файлами в папке sass
50.   gulp.watch('app/*.html', browserSync.reload); // Наблюдение за HTML файлами в корне проекта
51.   gulp.watch('app/js/**/*.js', browserSync.reload); // Наблюдение за JS файлами в папке js
52. });
53.
54. gulp.task('clean', function() {
55.   return del.sync('dist'); // Удаляем папку dist перед сборкой
56. });
57.
58. gulp.task('img', function() {
59.   return gulp.src('app/img/**/*.*) // Берем все изображения из app
60.     .pipe(cache(imagemin({ // Сжимаем их с наилучшими настройками с учетом кеширования
61.       interlaced: true,
62.       progressive: true,
63.       svgoPlugins: [{removeViewBox: false}],
64.       use: [pngquant()]
65.     })))
66.     .pipe(gulp.dest('dist/img')); // Выгружаем на продакшен
67. });
68.
69. gulp.task('build', ['clean', 'img', 'sass', 'scripts'], function() {
70.
71.   var buildCss = gulp.src([ // Переносим библиотеки в продакшен
72.     'app/css/main.css',
73.     'app/css/libs.min.css'
74.   ])
75.     .pipe(gulp.dest('dist/css'))
76.
77.   var buildFonts = gulp.src('app/fonts/**/*.*) // Переносим шрифты в продакшен
78.     .pipe(gulp.dest('dist/fonts'))
79.
80.   var buildJs = gulp.src('app/js/**/*.*) // Переносим скрипты в продакшен
81.     .pipe(gulp.dest('dist/js'))
82.
83.   var buildHtml = gulp.src('app/*.html') // Переносим HTML в продакшен
84.     .pipe(gulp.dest('dist'));
85.
86. });
87.
88. gulp.task('clear', function () {
89.   return cache.clearAll();
90. })
91.
92. gulp.task('default', ['watch']);
```

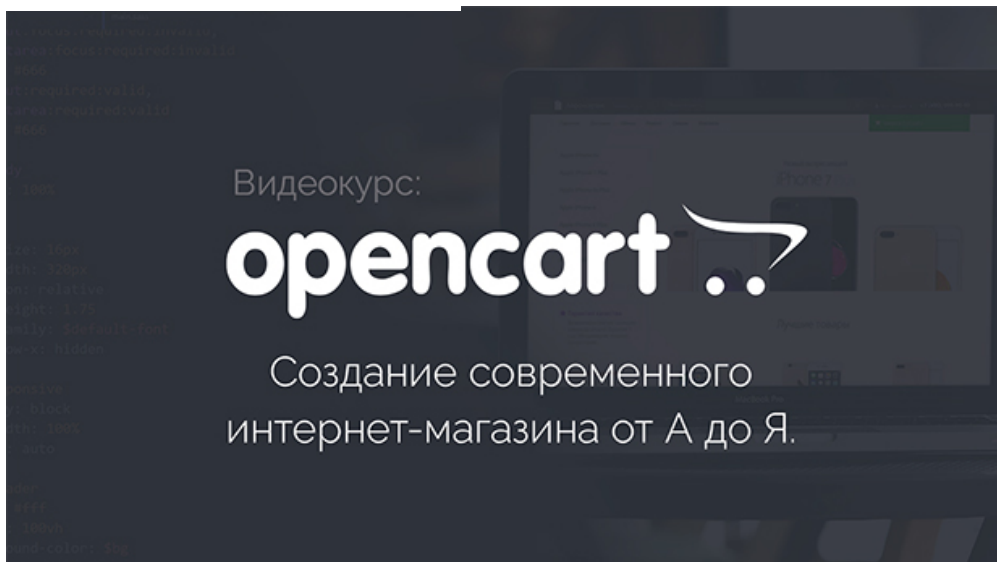
Проект-пример из данного урока вы можете посмотреть на GitHub и скачать: <https://github.com/agragregra/gulp-lesson>

Чтобы установить все пакеты и зависимости для скачанного примера, выполните команду **npm i** в папке проекта.

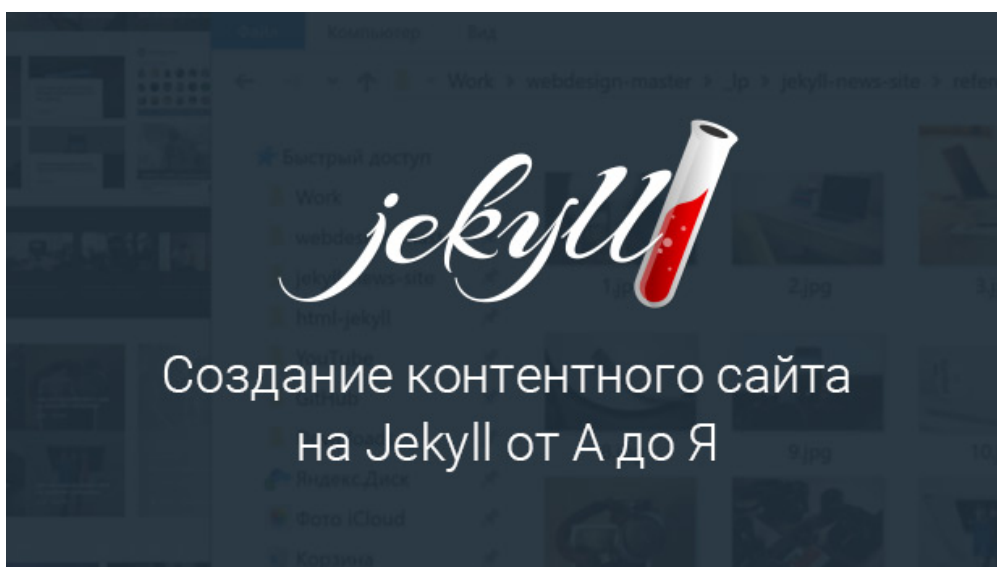
На этом все. Помните - к любому плагину для Gulp есть хорошая документация по подключению и использованию на npmjs.com или на страничке GitHub. Спасибо за внимание, друзья!

Премиум уроки от WebDesign Master:

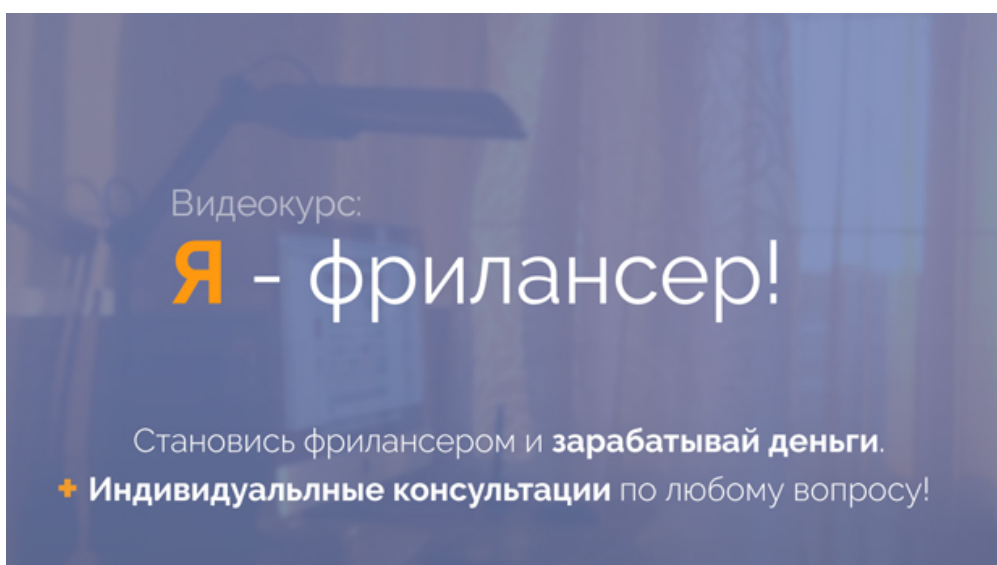




Создание современного интернет-магазина от А до Я



Создание контентного сайта на Jekyll от А до Я



Я - фрилансер! - Руководство успешного фрилансера

Google поиск по сайту...



2019 © WebDesign Master

