



Установка SASS

Вложенные правила

Переменные SASS

Директива @import

Директива @extend

Установка SASS

В [официальной документации](#) содержатся разные инструменты и предложения по компиляции SASS файлов (мы будем рассматривать SCSS синтаксис). При этом какой инструмент предпочтительнее и четкой последовательности действий вы там к сожалению не найдете. Сейчас в мире веб-разработки наибольшее количество разработчиков собирают свои проекты при помощи node.js и gulp. Почему так происходит?

Специальные приложения (prepros, koala, codeKit) можно использовать если вы начинающий разработчик, но как правило в крупных компаниях разработка осуществляется на сервере компании. Работать с ним вы будете через командную строку, и данные программы вы там физически запустить не сможете.

Если вы попробуете компилировать SASS файлы при помощи "ruby on rails", то столкнетесь с проблемами, если путь к компилируемому файлам содержит русские буквы. Ко всему прочему этот способ мягко говоря медленный. Как правило при верстки собираются не только стили, но и JavaScript-файлы. На данный момент времени лучшего способа собрать JS чем "webpack" просто не существует. По нормальному связать "webpack" и "ruby on rails" у вас не получится.

Перед началом установки, я рекомендую вам установить какой-нибудь файловый менеджер. Я использую [double commander](#). Могю сказать, что он бесплатный, кроссплатформенный, существует его портативная версия. С ним работать будет проще, но можно его и не устанавливать.

Итак, приступим к установке.

- Для начала идем на [официальный сайт node.js](#) ([Рис. 1](#)) и скачиваем его дистрибутив. Качаем его LTS-версию, иначе некоторые плагины могут не запуситься. После того, как мы его усановили, стоит проверить работает ли он. Для этого ждем "Win+I" (если у вас windows), в появившемся окне набираем "node" ([Рис. 2](#)), и ждем "Enter". Если все прошло успешно, то должно появиться вот такое окно ([Рис. 3](#)). Если вы начинающий разработчик, то можете использовать его как калькулятор, хотя node.js это полноценный сервер, не хуже чем "apache". Если же он не появился, то скорее всего у вас какие-то проблемы с правами. В операционных системах Windows Vista и Windows 7-8 нужно запускать инсталлятор с правами администратора.
- Далше идем на мою страничку [github](#) и качаем мой пример, если вы не умеете пользоваться git-ом, то вы можете скачать его ZIP-архив ([Рис. 4](#)). После того как вы скачали данный пример, его нужно где-нибудь распаковать. Поскольку пример учебный, я его распакую на диске "C:\\" ([Рис. 5](#)). Вам советую сделать так-же, в последующем имеет смысл создать папку для проектов, к примеру "projects" и все ваши проекты класть туда.

- Теперь поставим сборщик проектов gulp, и некоторые плагины к нему. Я рекомендую вам посмотреть [документацию по gulp](#). Из [документации следует](#), что сперва нам нужно установить "gulp-cli", а затем и сам "gulp". В моем примере нужно будет установить только "gulp-cli", сам сборщик проектов gulp, и все плагины для него установятся автоматически. Запустите "double commander" и перейдите в корень нашего проекта (в распакованный ранее архив из 2 пункта выше). Когда вы в него перейдете, то запустите в данном проекте терминал. Сделать это можно щелкнув на соответствующую иконку, она вторая слева ([Рис. 6](#)). Если вы не устанавливали "double commander", то запустите консоль и перейдите в наш проект. С "double commander" работать с консолью будет значительно проще (будет проще запустить консоль в нужном месте).

Далше в консоли следует набрать "npm install --global gulp-cli", начнется процесс установки. Все будет зависеть от скорости интернета, но в среднем установка занимает не более 5 минут. После того как вы установите "gulp-cli" наберите в консоли команду "npm install", у вас запустится установка ([Рис. 7](#)). После установки вы должны увидеть что-то вроде этого ([Рис. 8](#)). В проекте появится папка "node_modules", в ней лежат плагины которые будут компилировать наши SASS файлы.

Могю вас поздравить установка завершена. Давайте проверим, что у нас получилось.

- Перейдите в наш проект, запустите в нем консоль, и введите в консоли команду "gulp". У вас должно появиться вот такое вот окошко ([Рис. 9](#)). Это окошко говорит о том, что у нас запущился локальный сервер по адресу "http://localhost:8080/" и скомпилировались наши стили.
- В нашем проекте есть папка "scss", в ней лежит файл "main.scss". Он называется "сборочным-файлом" (еще я где-то читал он называется файлом-акселератором). В нем осуществляется подключение всех необходимых файлов. На выходе мы получим файл "main.css", который будет выводиться по следующему пути "public/css/main.css", так же мы будем компилировать его скатую версию по адресу "public/css/min/main.css". Наша сборка автоматически будет подставлять в выходном CSS "автопрефиксы", и будет группировать медиазапросы. То, что у нас отобразится в браузере будет лежать в папке "public". Я делал протестный пример.
- Итак перейдите по адресу <http://localhost:8080/>.
- Откройте файл "_example.scss" из папки "scss" в каком-нибудь редакторе. Я советую вам пользоваться Visual Studio Code, sublime text, или atom. Тут как говорится выберите сами.
- В данном файле у нас находится четыре переменных \$mainFontSize, \$mainBg, \$padding, \$margin. Переменная \$mainFontSize служит для задания размера шрифта нашего документа. Переменная \$mainBg служит для задания фона документа. Переменная \$padding служит для задания внутренних отступов слева, и справа нашего документа. Переменная \$margin служит для задания внешних отступов у абзацев нашей страницы. Впринципе это нужно нам для примера, поэтому усложнять не имеет смысла, суть будет ясна. Давайте попробуем поменять наши переменные.
- Замените переменную \$mainFontSize на 20px. Посмотрите как увеличился текст на странице, вместе с ним должны увеличиться и все отступы ([Рис. 10](#)).
- Для #wrapper замените параметр #darken(\$mainBg, 30%) на #darken(\$mainBg, 80%). Посмотрите как изменится текст документа ([Рис. 11](#)). Все изменения происходят в режиме реального времени. Нам не нужно перезагружать браузер. Впринципе компилятор мы настроили, дальше экспериментируйте с переменными сами.
- А теперь давайте попробуем *сломать* наш проект. В файле "_example.scss" замените переменную \$margin на \$margins, мы должны увидеть следующе окошко [Рис. 12](#). Оно нам говорит, что в строке 25 объявлена не используемая переменная. Очень важно уметь читать данные ошибки. Впринципе все, дальше попробуйте сами поменять переменные. Положите проект. Посмотрите, что будет.
- Для того, чтобы прекратить сборку, щелкните мышкой на консоль, и нажмите "Ctrl"+"C". Далее введите " ".

Принцип работы

Вроде бы все у нас получилось, установка прошла успешно, SASS файлы компилируются, локальный сервер настроен, при ошибках наш сборщик показывает номера строк с ошибками, но у вас наверно возник вопрос. А как все это работает?

Смотрите, в моем примере есть 2 файла: "package.json" и "gulpfile.js". Давайте их разберем более детально. В одном файле хранятся зависимости проекта, а в другом такие с заданкаи.

```
Package.json
1 {
2   "name": "sass-example",
3   "version": "0.0.0",
4   "description": "compile sass-file",
5   "main": "gulpfile.js",
6   "scripts": {
7     "test": "start"
8   },
9   "repository": {
10     "type": "git",
11     "url": "no"
12   },
13   "keywords": [
14     "compile",
15     "sass"
16   ],
17   "author": "maksim zhuchkov",
18   "license": "gnu",
19   "devDependencies": {
20     "gulp": "^4.0.2",
21     "gulp-autoprefixer": "^7.0.4",
22     "gulp-clean-css": "^4.2.0",
23     "gulp-connect": "^5.2.0",
24     "gulp-group-css-media-queries": "^1.2.2",
25     "gulp-sass": "^4.0.2"
26   },
27   "dependencies": {}
28 }
```

Файл package.json представляет собой объект javascript. Этот объект содержит служебную информацию с нашими зависимостями. Все параметры описывать я смысла не вижу, поэтому опишу самые основные.

Name - имя нашего проекта, в моем случае это [sass-example](#).
Version - версия нашей сборки. Так как я ничего не менял, то версия у меня 0.0.0. Если мы делаем не значительные изменения, то меняется последняя цифра. Если добавляется новый функционал, но сохраняется совместимость, то меняется вторая цифра. Если наша сборка становится не совместима с нашими прошлыми версиями, то меняется первая цифра.
Scripts - вообще, тут пишутся скрипты которые нужно запускать через NPM, но мы тут ничего писать не будем, так как смысла нет. Наша сборка собирается если в консоли набрать "gulp", эта запись коротче. В данном пункте как правило что-то добавляется, если мы будем использовать webpack, а его использование выходит за рамки данной статьи.
Author - эта ваш почерный слуга. Можете написать тут ваше имя.
License - так как мы являемся яркими сторонниками свободного ПО, то лицензия у нас будет gnu.
DevDependencies - здесь записаны те зависимости, которые нужны для сборки проекта. Те плагины, которые компилируют наши SASS файлы, снимают их, позволяют запустить локальный сервер. Более подробно эти плагины я опишу ниже. Знак ^^^ перед номерами версий говорит о том, что при установке стоит использовать пакеты данных или более поздних версий.
Dependencies - здесь записаны те зависимости, которые нужны нашему проекту. Те зависимости, которые войдут в итоговую сборку. Вообще проще так объяснить, если бы мы использовали webpack, то сюда были бы записаны: jquery, gaejs, swiper.js и другие плагины и библиотеки которые бы мы использовали. У нас простой пример, тут это не нужно. А так имейте ввиду.
Как я выше писал, файл [package.json](#) нужен для управления зависимостями. Представте, что вам нужно запустить проект со всеми зависимостями на другой машине. Без этого файла, вам бы пришлось, все модули, все библиотеки переносить на флешку. А так достаточно одного этого файла. Кидаем его в папку с проектом. Запускаем там консоль, и если у вас установлен node.js, то вам достаточно войти в эту папку через консоль и набрать в ней "npm install". Все зависимости подтянутся автоматически.

```
Gulpfile.js
1 // Выводим служебные функции gulp
2 const {src, dest, parallel, watch, series} = require('gulp');
3 // Объясняем наши модули
4 const sass = require('gulp-sass');
5 const connect = require('gulp-connect');
6 const autoprefixer = require('gulp-autoprefixer');
7 const gcmq = require('gulp-group-css-media-queries');
8 const cleanCSS = require('gulp-clean-css');
9 sass.compiler = require('node-sass');
10
11 // Настраиваем локальный сервер, локальный сервер будет брать файлы из папки public, и отображать их по адресу http://localhost:8080/
12 function serveTask(done) {
13   connect.server({
14     root: 'public',
15     livereload: true,
16     port: 8080,
17   }, function() {
18     this.server.on('close', done)
19   });
20 }
21
22 // Настраиваем задачу для компиляции стилей. Компилируем данный файл "scss/main.scss", в нем как правило подключаются остальные файлы. Пройдем тут автопрефиксер, группировку медиазапросов, минификацию файлов, и сделаем так, чтобы наша сборка показывала строку с ошибкой если мы ее допустим.
23 function style() {
24   return src('scss/main.scss')
25     .pipe(sass({on:'error', sass.logError}))
26     .pipe(autoprefixer({
27       cascade: false,
28       overrideBrowserslist: ['last 2 versions']
29     }))
30     .pipe(gcmq())
31     .pipe(dest('public/css'))
32     .pipe(cleanCSS({
33       level: 1
34     }))
35     .pipe(dest('public/css/min'))
36     .pipe(connect.reload());
37 }
38
39 // Просмотр файлов, если в папке "scss" происходят какие либо изменения с файлами имеюшие расширения ".scss" то выполняем задачу "style"
40 function watcher() {
41   watch(['scss/**/*.scss', 'style']);
42   watch(['scss/**/*.scss', 'style']);
43 }
44 exports.style = style;
45
46 // Задачи которые будут выполнены когда мы введем в консоли gulp. Будут скомпилированы стили, запущен watcher и локальный сервер. Все эти задачи будут запущены параллельно, независимо друг от друга.
47 exports.default = parallel(style, watcher, serveTask);
```

Gulpfile.js - это тот файл в котором хранятся наши "задачи". Он осуществляет управление gulp-плагинами. Он запускает локальный сервер, компилирует наши SASS файлы. Впринципе в нем все достаточно хорошо описано, позволять я опишу как он работает более детально.

В первой строчке мы выводим служебные функции gulp-a. Ниже я опишу, что они делают своими словами.

Src - условно можно сказать, что это точка входа, реке может быть путем входа. В нашем примере это "scss/main.scss". Более детально я это опишу когда дойдем до задачи.
Dest. Если функция src - это точка входа, то функция **dest** - это путь выхода. В нашем примере это "public/css" и "public/css/min" для скатой версии CSS (по этим путям будут компилироваться CSS-ки).
Parallel - функция которая будет выполнять задачи параллельно друг-другу. Появилась в 4 версии gulp-a. До этого все задачи выполнялись последовательно.
Series - функция которая будет выполнять задачи последовательно (друг за другом). Сперва выполнится первая задача, следом за ней вторая и тд. В нашем примере это не нужно. Обычно применяется в тех случаях, когда выполнение какой-либо задачи зависит от выполнения другой задачи. К примеру одна задача сжимает картинку, а вторая должна сжатье картинк перенести из одной папки в другую. Такие задачи должны быть выполнены этой функцией.
Watch - функция gulp-a которая осуществляет просмотр директорий. Просматриваемые директории задаются первым параметром, "пользовательские задачи" задаются вторым параметром. Как только в данных директориях происходит изменения, выполняется "пользовельский task".

С 4 по 9 строчку идет подключение gulp-плагинов. Эти плагины будут осуществлять компиляцию и минификацию наших SCSS-файлов, запускать локальный сервер. Gulp будет искать эти плагины в папке "node_modules", которая лежит на одном уровне с "gulpfile.js". Если он их не найдет, то поднимется на уровень выше, и так далее, пока не выйдет в корень диска.

Gulp-sass - плагин который компилирует SASS файлы.
Gulp-connect - локальный сервер. Я его ниже более детально опишу.
Gulp-autoprefixer - расставляет автопрефиксы для разных браузеров. Я обычно ставлю для 2 последних версий браузеров.
Gulp-group-css-media-queries - плагин который группирует наши медиа запросы. В реальных проектах мы можем в наш сборочный файл подключить разные SASS файлы с медиазапросами. Этот плагин возьмет из этих файлов медиазапросы, сгруппирует их, и вставит в конец нашего выходного CSS. У нас пример простой, нам это не нужно. Если вам это интересно посмотрите документацию. Создайте в папке "scss" папку "components", в ней создайте какие-нибудь SCSS файлы с медиазапросами (одинаковыми и разными), и подключите эти файлы в файле "main.scss" (если вы не умеете подключать SASS файлы ознакомьтесь с этой [статьей](#)). Посмотрите что получится.
Gulp-clean-css - плагин который осуществляет минификацию нашего CSS. Имеет разные параметры. Я точно не помню, но если поставить level: 2, или level: 3, то он не просто сожмет CSS, но еще и сгруппирует наши стили. Грубо говоря у нас объявлен 2 раза класс с разными свойствами. С этими параметрами он делает один класс и перенесет туда эти свойства.
С 12 по 20 строчку настраивается локальный сервер. Он работает на порту "8080", а файлы которые будут показаны в браузере он будет брать из папки "public". Наш локальный сервер будет поддерживать "livereload". Livereload - это такая технология, которая будет перезагружать браузер, когда мы сделаем изменения после компиляции нашей CSS-ки. В нашем коде перезагрузка браузера вызывается вот так "pipe(connect.reload())".

С 23 по 37 строчку описывается задача которая компилирует SCSS файлы. На самом деле, в этой задаче компилируется только один файл "main.scss". Давайте более детально это разберем.
Return src('scss/main.scss') - каждый "пользовательский task" должен что-то возвращать. Поэтому он начинается с "return". Далее мы говорим взять файл "main.scss" (src('scss/main.scss')).
Pipe(sass({on:'error', sass.logError})) - здесь мы говорим компилятору применить к файлу который мы взяли ранее, плагин "gulp-sass". Впринципе, у нас уже есть CSS-ка, но она пока хранится в оперативной памяти. Как вы помните, она будет вставлена когда наш компилятор увидит функцию dest(). Вызывая плагин с этими параметрами (error, sass.logError), мы просим наш компилятор выводить ошибки, если мы их допустим.

Pipe(autoprefixer(...)) - этой командой мы говорим нашему компилятору применить к нашей CSS-ки плагин "gulp-autoprefixer". У меня для него стоит стандартные параметры для 2 последних версий браузеров (last 2 versions). Поскольку наша CSS-ка находится в оперативной памяти, изменения происходят быстро.

Pipe(gcmq()) - этой командой мы говорим нашему компилятору применить к нашей CSS-ки плагин "gulp-group-css-media-queries". Как я выше писал, этот плагин возьмет все медиазапросы, сгруппирует их, и вставит в самый конец нашей CSS-ки. Это очень удобно. Поскольку наша CSS-ка все еще находится в оперативной памяти, эта операция выполнится быстро.

Pipe(dest('public/css')) - этой командой мы говорим нашему компилятору вставить нашу CSS-ку из оперативной памяти, в конечный файл по адресу "public/css". Имя вставляемой CSS-ки будет такое-же как и имя нашего файла на входе (src('scss/main.scss')) соответственно получится main.css).

Pipe(cleanCSS(...)). Наша задача еще не окончена. CSS-ка все еще есть в оперативной памяти, и этой командой мы говорим применить к ней плагин "gulp-clean-css". Проще говоря этот плагин ее сожмет. У него есть много параметров сжатия, я выбираю "level: 1", то есть просто ее сжимаю. Большого мне не нужно.

Pipe(dest('public/css/min')) - данной командой мы ее вставляем по адресу "public/css/min".

Pipe(connect.reload()) - данной командой мы говорим перезагрузить браузер в котором открыт наш проект. Локальный сервер настроен на порт "8080", поэтому мы можем его перезагрузить. Это очень удобно. Практически это все равно, что вставить в "filebug-e". Особенно круто, если вы встаете "perfect pixel". Поскольку наша задача закончилась, CSS-ка из оперативной памяти удалится.

С 40 по 43 строчку написан простой "watcher". Этой функцией мы говорим просматривать папку "scss" (в коде об этом говорит такая запись - "scss/*.scss"), и все ее подпапки ("scss/**/*.scss"). Просматривать мы будем только файлы с расширением ".scss". Вторым параметром мы передаем наш "task", который компирует сборочный SASS файл. Как он работает я описал выше. При любом изменении SASS файлов произойдет сборка CSS-ки. Впринципе, если написать "scss/**" вместо "scss/**/*.scss" то компиляция будет происходить при любом изменении в папке "scss". Вообще принято указывать расширения при которых будет происходить компиляция, но мало кто это делает. Ошибки тут нет, каждый пишет, как привык.

В 47 строчке написано какие задачи будут выполнены, когда мы в консоле напишем "gulp" (об этом говорит такая запись - exports.default). Наши задачи будут запущены параллельно друг-другу. По этой строчке видно, что мы компилируем наши CSS-ки, запускаем watcher и локальный сервер.

Не много про архитектуру проекта

Как я выше писал, я сделал пример достаточно простой. Но даже в таком простом примере папка "node_modules" весит 25 Мб. Вообще это много. Если вы только верстаете, то скорее всего у вас все верстка будет весить меньше. Если так в каждый проект ставить зависимости, то никакого жесткого не хватит. Если вы еще и "babel" к вашему проекту прикрутите, то места точно не хватит.

Я рекомендую вам удалить папку "мои проекты" (на каком диске ее deletes решайте сами, это не важно, название папки точно не важно, главное суть). Внутри сделать еще одну папку, к примеру "project". Внутри этой папки закиньте файл package.json и установите gulp-плагины туда. На этом уровне стоит создавать папки с проектами. К примеру создаи файл "интернет магазин 1" (тут подразумеваем название вашего проекта, то, что вы будете верстать), в нее стоит закинуть gulp файл. Когда вы запустите "gulp" из командной строки, то он не найдет описываемых в нем плагинов, поднимется на уровень выше и найдет их там. Таким образом вы успешноно съэкономили место. Если плагины от проекта удали и теже, нет никакого смысла ставить их в каждый проект. Когда в папке "projects" наберется много проектов, и вам будет сложно в ней ориентироваться, или появится новая версия "gulp-a", то возьмите папку "projects", и переименуйте ее в папу "projects_old_год.переименования". Создайте новую папку "projects", установите туда нужные вам gulp-плагины и работайте в ней. Я впервые с NPM познакомился в 2014 году, тогда я еще сборки делал на "grunt-e", и у меня папок с проектами уже очень много набралось. Будете переименовывать папку "projects" обязательно ставьте под переименования, думайте о будущем.

Иногда так бывает, что создать новую папку "projects" нельзя. Дома вы можете делать как угодно, а вот на работе так не получится. На моей прошлой работе так было делать нельзя. Смотрите, когда мы пишем в "gulpfile.js" - "const sass = require('gulp-sass')", то "gulp" будет искать этот gulp-плагин на этом же уровне, в папке "node_modules". Как вы помните, если на этом уровне он gulp-плагины не найдет, то он поднимется на уровень выше, и будет искать их там, и тд. Но ведь мы можем и жестко задать пути до gulp-плагинов. На моей прошлой работе у нас была папка с проектами, выше лежала папка "node_modules", когда вышла новая версия "gulp-a", мы выше чем наша папка с проектами создали папку с новыми gulp-плагинами, установили их туда и жестко прописали в "gulpfile.js" путь к ним. Получилось что-то вроде этого, ("const sass = require('.../gulp_plugin_19/node_modules/gulp-sass')). Вот такая запись ".../.." говорит подняться на 2 уровня вверх (уровень папки с проектом, и еще на один уровень вверх), а дальше идет путь до наших плагинов. Так можно будет совмещать старые и новые версии "gulp-a".

Вместо заключения

Я старался максимально подробно описать процесс установки и настройки SASS. Если у вас есть какие-то вопросы, пожелания, или у вас что-то не получается, то вы можете спросить меня по электронной почте, или написать мне вконтакте. Я с радостью вам помогу. Впринципе если вы внимательно ознакомитесь с этой статьей, то легко сможете разобраться в любом gulp файле. Это не трудно. Пример который я тут описал достаточно простой. В реальной жизни собираются не только стили, но и HTML-шаблоны, JS. При помощи "gulp-a" вы можете сделать архив вашего проекта, проверить на валидность ваши HTML-файлы и многое-многое другое. Если интересно как это происходит посмотрите под [этот проект](#), я стараюсь его развивать по мере возможности. Для того, чтобы посмотреть какие есть плагины для "gulp-a" посмотрите под [эту страничку](#), здесь есть плагины на любой вкус. Как их установить достаточно хорошо описано, главное помнить служебные функции "gulp-a", и знать, что они делают (напомню это - src(), dest(), parallel(), watch(), series()).

