МИНИСТЕРСТВО НАУКИ и высшего образования РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

филиал «РКТ» МАИ в г. Химки Московской области

**Специальность 09.02.03 «Программирование в компьютерных системах»**

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**ПМ.02«Разработка и администрирование баз данных**»

**Студент**

**Группы МП-32** Славгородский К. В. **/(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**

**Руководитель**

**практики от организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**

**Руководитель**

**практики от филиала** Шумаев А.Ю. **/(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**

**Химки**

**2020г.**

**ПРОГРАММА ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

По специальности 09.02.03 «Программирование в компьютерных системах»

|  |  |  |
| --- | --- | --- |
| Дата | Наименование выполняемых работ | Количество дней практики |
| 11.06.20 | Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. | 4 |
| 12.06.20 | Описание структуры сети, в которой может функционировать разработанная база данных. | 4 |
| 13.06.20 | Описание сетевого оборудования необходимого для работы с базой данных. | 4 |
| 15.06.20 | Описание программного обеспечения необходимого для работы с базой данных по сети. | 4 |
| 16.06.20-18.06.20 | Разработка клиентских интерфейсов для клиент-серверных приложений. | 16 |
| 20.06.20 | Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. | 4 |
| 22.06.20-24.06.20 | Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. | 16 |
| 25.06.20-01.07.20 | Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. | 50 |
| 02.07.20-03.07.20 | Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. | 12 |
| 03.07.20-04.07.20 | Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. | 12 |
| 06.07.20 | Составление отчёта по практике | 6 |
| 07.07.20 | Подготовка к защите отчёта по практике. | 6 |
| 08.07.20 | Итоговая аттестация по производственной практике - зачёт | 6 |

Руководитель практики от филиала «РКТ» МАИ преподаватель

Шумаев А.Ю.

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

**АТТЕСТАЦИОННЫЙ ЛИСТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Славгородский Кирилл Васильевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*ФИО*

обучающийся на 3-ем курсе по специальности СПО 09.02.03 «Программирование в компьютерных системах»

успешно прошел производственную практику по профессиональному модулю   
**ПМ.02 «Разработка и администрирование баз данных»**

в объеме 144 часа с 11.06.2020г. по 08.07.2020г.

в организации филиала «РКТ» МАИ

Виды и качество выполнения работ

|  |  |  |
| --- | --- | --- |
| Виды работ, выполненных обучающимся во время практики | Объем работ | Качество выполнения работ в соответствии с технологией и (или) требованиями организации, в которой проходила практика |
| Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. | 4 |  |
| Описание структуры сети, в которой может функционировать разработанная база данных. | 4 |  |
| Описание сетевого оборудования необходимого для работы с базой данных. | 4 |  |
| Описание программного обеспечения необходимого для работы с базой данных по сети. | 4 |  |
| Разработка клиентских интерфейсов для клиент-серверных приложений. | 16 |  |
| Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. | 4 |  |
| Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. | 16 |  |
| Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. | 50 |  |
| Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. | 12 |  |
| Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. | 12 |  |
| Составление отчёта по практике | 6 |  |
| Подготовка к защите отчёта по практике. | 6 |  |
| Итоговая аттестация по производственной практике - зачёт | 6 |  |

Руководитель практики от филиала «РКТ» МАИ Шумаев А.Ю.

|  |  |  |
| --- | --- | --- |
| Дата\_\_\_\_\_\_\_\_\_\_\_\_\_ | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *подпись* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Расшифровка подписи* |

**ДНЕВНИК** **ПРОХОЖДЕНИЯ ПРАКТИКИ**

|  |  |  |
| --- | --- | --- |
| Дата | Наименование выполняемых работ | Подпись  руководителя |
| 11.06.20 | Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. |  |
| 12.06.20 | Описание структуры сети, в которой может функционировать разработанная база данных. |  |
| 13.06.20 | Описание сетевого оборудования необходимого для работы с базой данных. |  |
| 15.06.20 | Описание программного обеспечения необходимого для работы с базой данных по сети. |  |
| 16.06.20-18.06.20 | Разработка клиентских интерфейсов для клиент-серверных приложений. |  |
| 20.06.20 | Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. |  |
| 22.06.20-24.06.20 | Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. |  |
| 25.06.20-01.07.20 | Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. |  |
| 02.07.20-03.07.20 | Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. |  |
| 03.07.20-04.07.20 | Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. |  |
| 06.07.20 | Составление отчёта по практике |  |
| 07.07.20 | Подготовка к защите отчёта по практике. |  |
| 08.07.20 | Итоговая аттестация по производственной практике - зачёт |  |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc54044402)

[1 GITHUB 8](#_Toc54044403)

[1.1Распределённая система управления версиями. 8](#_Toc54044404)

[1.2 Преимущества Git 8](#_Toc54044405)

[1.2 Веб-сервис для хостинга и совместной разработки 9](#_Toc54044406)

[1.3 Сервер базы данных (SQL-сервер) 10](#_Toc54044407)

[1.4 Функционирование сервера базы данных 10](#_Toc54044408)

[1.5 Локальные системы контроля версий 11](#_Toc54044409)

[1.6 Централизованные системы контроля версий 11](#_Toc54044410)

[1.7 Распределённые системы контроля версий 12](#_Toc54044411)

[2 GULP 14](#_Toc54044412)

[Установка Gulp 16](#_Toc54044413)

[Создание проекта Gulp 17](#_Toc54044414)

[Структура каталогов в проектах 21](#_Toc54044415)

[gulpfile.js 22](#_Toc54044416)

[Gulp Sass 24](#_Toc54044417)

[Выборка файлов для gulp.src 26](#_Toc54044418)

[Наблюдение за изменениями в файлах (Gulp Watch) 27](#_Toc54044419)

[Автоматическое обновление страниц с использованием Bbrowser Sync 28](#_Toc54044420)

[Оптимизация JavaScript 33](#_Toc54044421)

[Подготовка к продакшену 37](#_Toc54044422)

[Оптимизация изображений 39](#_Toc54044423)

[Автоматическое создание префиксов CSS с помощью Gulp 42](#_Toc54044424)

[Стандартный таск Gulp 43](#_Toc54044425)

[3 СИСТЕМА ОТСЛЕЖИВАНИЯ ОШИБОК 50](#_Toc54044426)

[ЗАКЛЮЧЕНИЕ 53](#_Toc54044427)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 54](#_Toc54044428)

# ВВЕДЕНИЕ

В современном мире, в условиях индустриализации, ежедневно растёт число избыточной информации, в которой иногда трудно сориентироваться и выбрать нужные сведения.

Для решения подобных проблем применяются автоматизированные базы данных. Они стали неотъемлемой частью практически всех компьютерных систем - от отрасли до отдельного предприятия. За последние несколько лет вырос уровень потребительских качеств систем управления базами данных (СУБД): разнообразие поддерживаемых функций, удобный для пользователя интерфейс, сопряжение с программными продуктами, в частности с другими СУБД, возможности для работы в сети и т.д. Серверная часть СУБД позволяет сводить воедино информацию из самых разных источников (электронные таблицы, другие базы данных) и помогает быстро найти необходимую информацию, донести ее до окружающих с помощью отчетов, графиков или таблиц.

Базы данных — это совокупность структур, предназначенных для хранения больших объемов информации и программных модулей, осуществляющих управление данными, их выборку, сортировку и другие подобные действия.

Реляционные базы данных получили широкое применение в различных предметных областях и различных архитектурах.

Gulp – это таск-менеджер для автоматического выполнения часто используемых задач, написанный на языке программирования JavaScript.

# 1 GITHUB

# Распределённая [система управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8).

[Git](https://git-scm.com/about) — распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками. Она была разработана в 2005 году Линусом Торвальдсом, создателем Linux, для того, чтобы другие разработчики могли вносить свой вклад в ядро Linux. Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами.

Системы контроля версий (СКВ, VCS, Version Control Systems) позволяют разработчикам сохранять все изменения, внесённые в код. Поэтому в случае, описанном выше, они могут просто откатить код до рабочего состояния вместо того, чтобы тратить часы на поиски маленькой ошибки или ошибок, ломающих весь код.

СКВ также дают возможность нескольким разработчикам работать над одним проектом и сохранять внесённые изменения, чтобы убедиться, что все могут следить за тем, над чем они работают.

Git стоит отдельно от других СКВ из-за подхода к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

# 1.2 Преимущества Git

* **Бесплатный и open-source**. Это значит, что его можно бесплатно скачать и вносить любые изменения в исходный код;
* **Небольшой и быстрый**. Он выполняет все операции локально, что увеличивает его скорость. Кроме того, Git локально сохраняет весь репозиторий в небольшой файл без потери качества данных;
* **Резервное копирование**. Git эффективен в хранении бэкапов, поэтому известно мало случаев, когда кто-то терял данные при использовании Git;
* **Простое ветвление**. В других СКВ создание веток— утомительная и трудоёмкая задача, так как весь код копируется в новую ветку. В Git управление ветками реализовано гораздо проще и эффективнее.
* Основные команды.
* Всего несколько команд нужно для базового варианта использования Git для ведения истории изменений.
* git add. Команда git add добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.
* git status. Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.
* git diff. Команда git diff используется для вычисления разницы между любыми двумя Git деревьями.
* git difftool. Команда git difftool просто запускает внешнюю утилиту сравнения для показа различий в двух деревьях, на случай если вы хотите использовать что-либо отличное от встроенного просмотрщика git diff.
* git commit. Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.
* git clean. Команда git clean используется для удаления мусора из рабочей директории. Это могут быть результаты сборки проекта или файлы конфликтов слияний.

# 1.2 [Веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для [хостинга](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3) и совместной разработки

[GitHub](https://github.com/) — сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше. Обычно он используется вместе с Git и даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах.

Также GitHub может похвастаться контролем доступа, багтрекингом, управлением задачами и вики для каждого проекта. Цель GitHub — содействовать взаимодействию разработчиков.

К проекту, загруженному на GitHub, можно получить доступ с помощью интерфейса командной строки Git и Git-команд. Также есть и другие функции, такие как документация, запросы на принятие изменений (pull requests), история коммитов, интеграция со множеством популярных сервисов, email-уведомления, эмодзи, графики, вложенные списки задач, система @упоминаний.

Git — это инструмент, позволяющий реализовать распределённую систему контроля версий, а GitHub — это сервис для проектов, использующих Git.

# 1.3 Сервер базы данных (SQL-сервер)

СЕРВЕР БАЗЫ ДАННЫХ (англ. database server), также SQL-сервер - сервер, обеспечивающий хранение, обработку и управление структурированными данными (база данных), а также обеспечивающий операции ввода-вывода при доступе клиентов к хранящейся информации. Сервер базы данных занимается обслуживанием базы данных и отвечает за целостность и сохранность хранимых данных. Технически, сервер базы данных представляет собой очень мощный сервер, либо кластер серверов или "ферму серверов", с установленным специальным программным обеспечением, осуществляющим систему управления базами данных (СУБД). Клиенты, которым необходимо прочитать или записать информацию в СУБД, осуществляют доступ к серверу базы данных по сети.

# Функционирование сервера базы данных

Основная идея сервера базы данных состоит в том, чтобы размещать, хранить и обрабатывать данные на сервере с мощными вычислительными ресурсами, а с помощью, установленной на нем системы управления базами данных (СУБД) предоставить приложениям, использующим специальные запросы к СУБД, быструю обработку этих запросов и высокоскоростной доступ. Для организации работы такой системы применяется архитектура клиент-сервер. Для управления современными базами данных как правило используется язык структурированных запросов SQL (англ. structured query language). Сервер, работающий под управлением СУБД, использующей язык SQL, называется SQL-сервер. В настоящее время наиболее распространены следующие программные продукты (SQL-серверы), реализующие систему управления базами данных (СУБД): Microsoft SQL Server; Oracle Database Server; IBM DB2; MySQL, PostgreSQL и другие.

# Локальные системы контроля версий

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.



Рисунок 1. Локальный контроль версий.

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

# 1.6 Централизованные системы контроля версий

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.



Рисунок 2. Централизованный контроль версий.

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы: когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

# 1.7 Распределённые системы контроля версий

Здесь в игру вступают распределённые системы контроля версий (РСКВ). В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.



Рисунок 3. Распределённый контроль версий.

Более того, многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

# 2 GULP

Gulp - это инструмент для автоматизации рутинных задач, которые возникают при веб-разработке. Это может быть не только frontend разработка, это может быть и backend разработка.

Если вы работаете с такими технологиями как html,  css, javascript и.т.д. Если вы внедрите в практику своей работы такой инструмент как gulp, вы значительно ускорите скорость вашей работы и, во-вторых, этот инструмент "откроет вам дорогу" к новым возможностям, которые значительно повысят ваш уровень веб-разработки и знаний.

Gulp - это просто программа, которая написана на языке программирования Javascript. Для того, чтобы начать его пользоваться, желательно знать, хотя бы основы языка Javascript. Если вы это знаете, то пользоваться программой Gulp для вас будет намного проще.

Смысл следующий: мы создаем для системы Gulp некие задания. Т.е. описываем эти задания на языке Javascript. Затем, Gulp просто выполняет эти задания в нужной последовательности, так, как мы это прописали. Т.е. Gulp - это просто система по управлению заданиями по веб-разработке. Также ее называют task manager. task - задание, manager - управлять.

Важно понимать, что Gulp - это просто некое ядро, к которому мы прикручиваем дополнительные модули, плагины, которые "учат" Gulp делать какую-то определенную функцию или работу. Устанавливая эти плагины мы получаем новые возможности в системе Gulp, которые мы можем использовать.

Если вернуться на официальную страницу сайта Gulp и перейдем в раздел Plugins, то здесь мы можем найти большое количество плагинов, которые позволяют нам выполнять какие-то определенные рутинные задачи.

Выбираем нужный плагин, устанавливаем его в эту систему и пользуемся им.

Какие типовые задачи можно решать с помощью этой системы?

Давайте рассмотрим самые основные из них. Список этих задач может быть достаточно большой.

**Задача 1. Минификация кода.**

Это одна из самых частых задач, для которой чаще всего используют Gulp и подобные ей системы - это задача минификации кода. Т.е. вы написали какой-то код, на каком-то языке программирования. Для вас этот код воспринимается хорошо, но, если вы разместите этот код на рабочем сервере, на котором будет размещаться ваш сайт, то, соответственно, этот код будет загружаться довольно долго из-за того, что в нем есть много лишней информации в виде отступов, комментариев и.т.д.

Gulp позволяет убрать все лишнее из кода, подготовить его для того, чтобы это можно было выложить на рабочий сервер.

**Задача 2. Объединение кода из разных файлов в один.**

Вы можете объединять код из CSS, Javascript файлов и.т.д. в один. Это важно сделать также по причине скорости загрузки документа. При работе с протоколом http каждый запрос к файлу - это дополнительное время загрузки страницы.

Если вы объединяете код в один файл, загрузить его проще и быстрее, чем загрузить несколько файлов.

Это типовая задача, которую приходится решать в современной веб-разработке. Написать программу намного проще, если ее код разбит на модули и независимые части.

**Задача 3. Работа с CSS препроцессорами: sass, less, …**

Система Gulp позволяет вам использовать их в своей работе и вы получите такой мощный инструмент для того, чтобы улучшить свои навыки веб-разработки.

**Задача 4. Поддержка новых стандартов языка Javascript.**

Т.к. язык Javascript является клиентским языком программирования, то он зависит того браузера, на котором будет работать. Если посетитель вашего сайта пользуется какими-то старыми браузерами, у него новые стандарты не будут работать. С помощью Gulp вы можете решить эту задачу.

Gulp - это не единственный инструмент, который позволяет решать такие подобные задачи. Важно понимать, что Gulp - это одно из самых простых и легких решений, которые позволяют это сделать.

Код написанный под Gulp интуитивно понятный и проект, который у вас будет получаться, достаточно компактный и удобный. В нем будет все самое необходимое, что нужно для веб-разработки.

Есть и другие решения, например, система Webpack, которая в последнее время приобретает большую популярность. Но, дело в том, что Webpack - это довольно громоздкая система и для решения каких-то простых задач устанавливать такую большую систему не очень разумно.

Gulp - это альтернатива, которая позволит вам решать эти задачи проще и быстрее.

**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 должен быть установлен Node.js. Установка Node.js для различных платформ довольно простая - скачиваете инсталлер Node для своей операционной системы и устанавливаете.

После того, как Node установлен, можно приступать к установке Gulp. Нужно открыть терминал (правая кнопка мыши в папке с зажатым Shift> Открыть здесь оболочку Linux) и выполнить следующую команду:

|  |
| --- |
| npm i gulp -g |

Для пользователей Mac и Linux и Ubuntu bash в Windows, глобальную установку с ключом -g необходимо выполнять с правами суперпользователя, **sudo**, например:  
**sudo npm i gulp -g**.

Из данной команды видно, 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.

### Создание проекта Gulp

Надо создать папку проекта для примера, папка **myproject**.

Важно! Нельзя создавать русскоязычные папки проектов и нужно следить за тем, чтобы путь до папки проекта не содержал кириллических символов, то-есть не был написан на русском языке. В противном случае, могут возникнуть проблемы при работе различных утилит Gulp. Папка 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 "Откройте здесь оболочку Linux". Оболочка Linux должна быть предварительно установлена.

Далее выполним инициализацию проекта в той папке, которую создали:

|  |
| --- |
| npm init |

Следуя инструкциям, заполним метаинформацию о проекте:

Называем проект "MyProject"

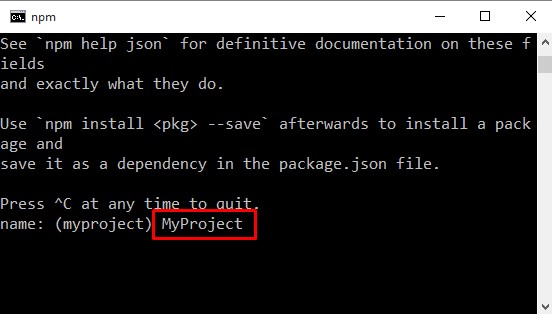


Рисунок 4

Оставляем версию текущей - 1.0.0

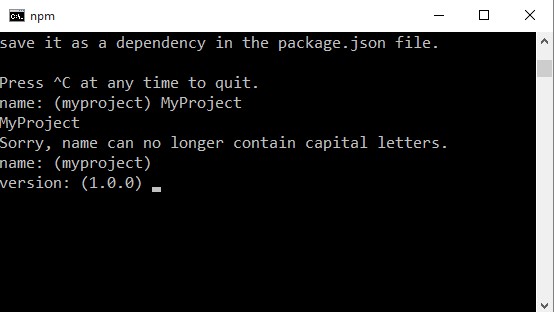


Рисунок 5

Введем краткое описание проекта - My First Gulp Project:

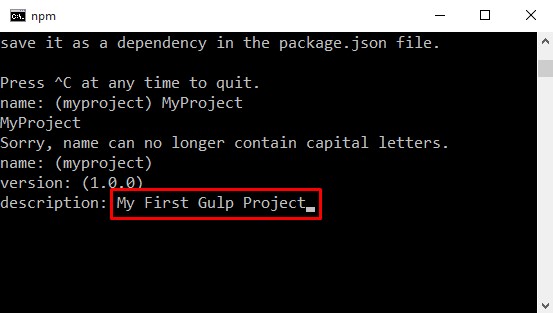


Рисунок 6

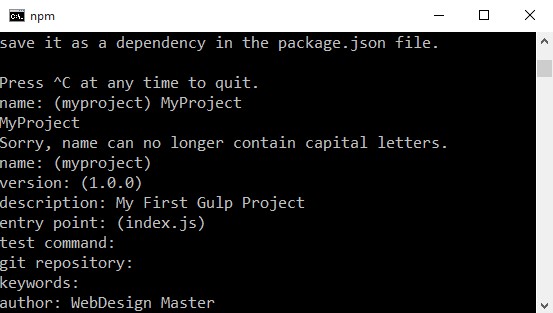
entry point, test command, git repository, keywords оставить по-умолчанию. 

Рисунок 7

license оставить по-умолчанию и ввести yes:

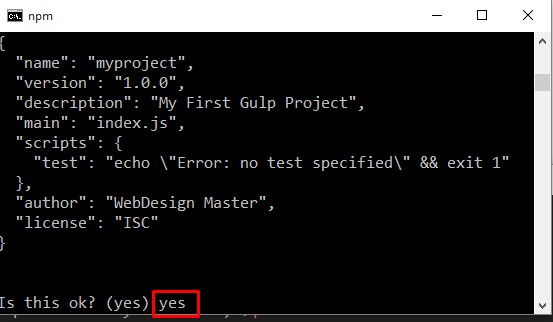


Рисунок 8

В результате такой несложной первоначальной настройки нового Gulp проекта в папке myproject появится новый файл **package.json**. Результат изображен на рисунке 9

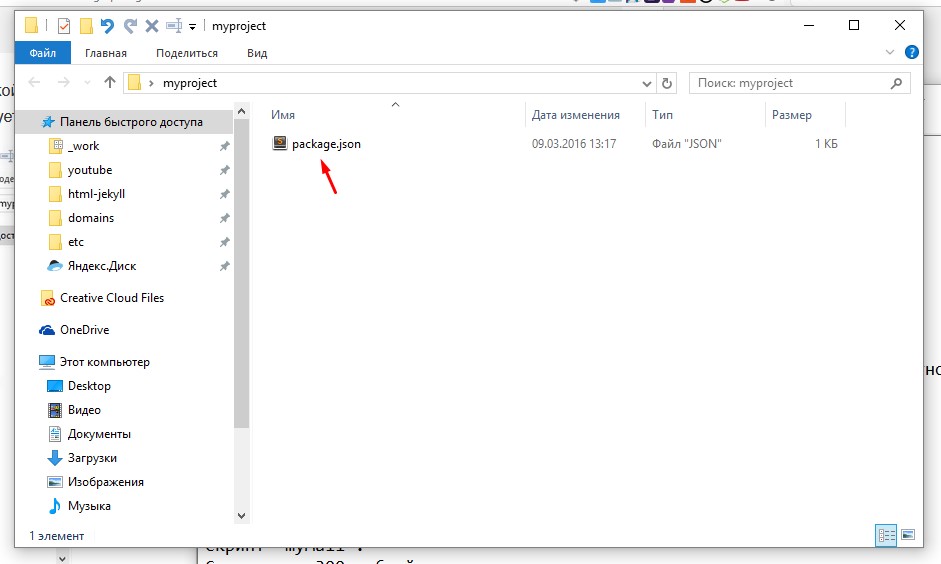


Рисунок 9-папка myproject

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

Например, если установить в проект Gulp с ключом **--save-dev**, то пакет и используемая версия автоматически добавится в 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.

Установка в проект Gulp:

|  |
| --- |
| npm i gulp --save-dev |

Видно, что 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:



Рисунок 10-установка Gulp

Кроме того, появляется папка **node\_modules**, которая теперь содержит установленный пакет gulp и необходимые зависимости. В данную папку автоматически будут сбрасываться все модули и зависимости, которые будут устанавливаться в проект. Папок с зависимостями может быть очень много, несмотря на то, что 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. Кроме того, может появиться дополнительный файл **package-lock.json**.

### 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, у всех проектов есть схожая структура каталогов, например, большинство проектов имеет папку **dist** и **app**. Создаем первоначальную структуру учебного проекта. В результате должны создать следующую структуру в проекте myproject

* **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 используется именно такая структура проекта.

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

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

### gulpfile.js

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

|  |
| --- |
| var gulp = require('gulp'); |

Данной строчкой подключаем Gulp к проекту, посредством функции **require**. Данная функция подключает пакеты из папки node\_modules в проект, присваивая их переменной. В данном случае, создаем переменную **gulp**.

Далее уже можно работать с этой переменной и создавать таски (инструкции).

|  |  |
| --- | --- |
|  | gulp.task('mytask', function() { |
|  | console.log('Привет, я таск!'); |
|  | }); |

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

|  |
| --- |
| gulp mytask |

На рисунке 11 изображен файл **gulpfile.js**:

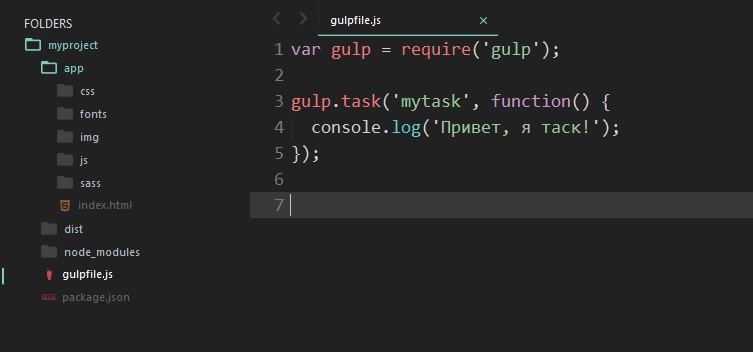


Рисунок 11-gulpfile.js

На рисунке 12 представлен результат выполнения команды **gulp mytask**:

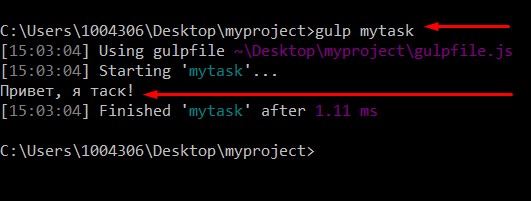


Рисунок 12- результат выполнения команды **gulp mytask**

Если используется Gulp 4 и появляется ошибка о невозможности завершения таска, можно добавить async перед function() и выполнять код асинхронно:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'); |
|  | gulp.task('mytask', async function() { |
|  | console.log('Привет, я таск!'); |
|  | }); |

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

|  |  |
| --- | --- |
|  | gulp.task('mytask', function () { |
|  | return gulp.src('source-files') *// Выборка исходных файлов для обработки плагином* |
|  | .pipe(plugin()) *// Вызов Gulp плагина для обработки файла* |
|  | .pipe(gulp.dest('folder')) *// Вывод результирующего файла в папку назначения (dest - пункт назначения)* |
|  | }) |

* **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 нужно создать обработчик, который будет компилировать Sass файлы в CSS (CSS препроцессинг).

### Gulp Sass

Устанавливаем пакет **gulp-sass** в проект с сохранением версии и названия в package.json.

Любые Gulp пакеты, для любых задач, легко находятся и имеют вполне исчерпывающие инструкции по подключению на своих домашних страницах и в документации.

|  |
| --- |
| npm i gulp-sass --save-dev |

Далее подключаем gulp-sass в файле gulpfile.js. Переменные для подключения пакетов можно перечислять через запятую:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), |
|  | sass = require('gulp-sass'); *//Подключаем Sass пакет* |

Создаем в папке app/sass файл **main.sass**, задаем в нем фон body - черный и пишем для него обработчик в gulpfile.js. На рисунке 13 изображен файл main.sass

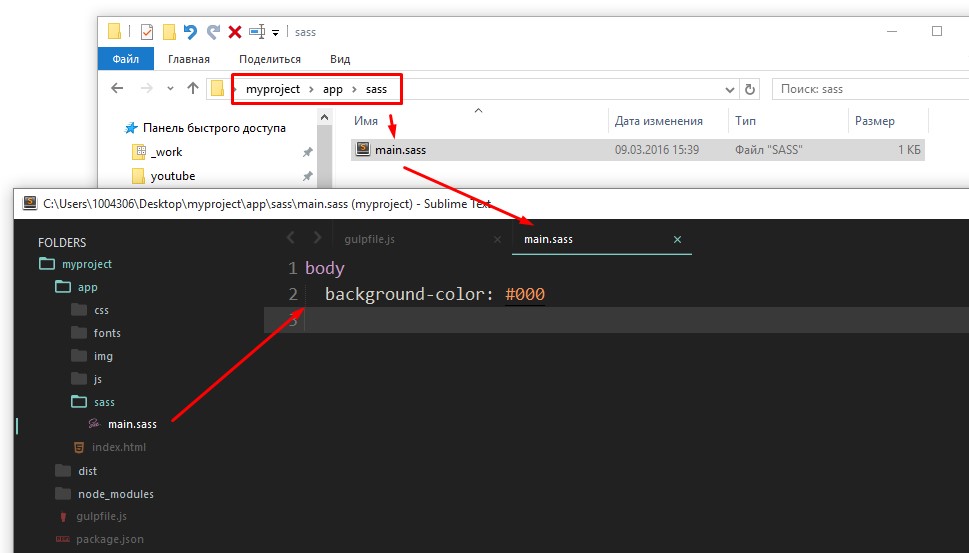


Рисунок 13 -main.sass

**gulpfile.js**:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'); *//Подключаем Sass пакет* |
|  |  |
|  | gulp.task('sass', function(){ *// Создаем таск "sass"* |
|  | return gulp.src('app/sass/main.sass') *// Берем источник* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(gulp.dest('app/css')) *// Выгружаем результата в папку app/css* |
|  | }); |

После этого, надо выполнить в терминале новый таск **sass**:

|  |
| --- |
| gulp sass |

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

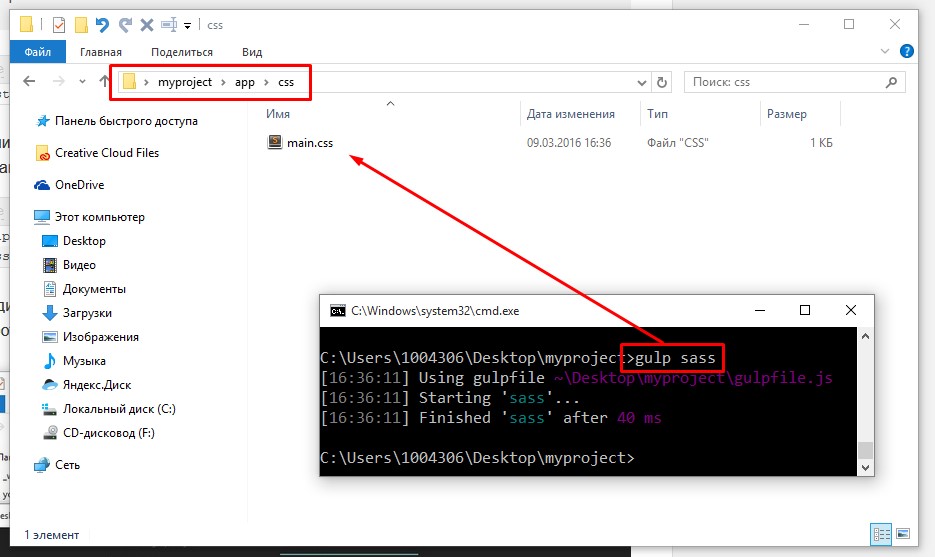


Рисунок 14-результат выполнения команды

### Выборка файлов для gulp.src

Было рассмотрено все, что необходимо знать о Gulp, теперь можно углубляться в каждую деталь того, что было изложено выше.

Выборка файлов в примере выше довольно простая, брали файл напрямую: **gulp.src('app/sass/main.sass')**. Но файлы также можно выбирать по шаблону. Шаблон выборки файлов называется **glob**.

#### 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.

Внесем некоторые изменения в таск **sass** и сделаем его более универсальным:

|  |  |
| --- | --- |
|  | gulp.task('sass', function(){ |
|  | return gulp.src('app/sass/\*\*/\*.sass') *// Берем все sass файлы из папки sass и дочерних, если таковые будут* |
|  | .pipe(sass()) |
|  | .pipe(gulp.dest('app/css')) |
|  | }); |

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. На рисунке 15 gulp sass.

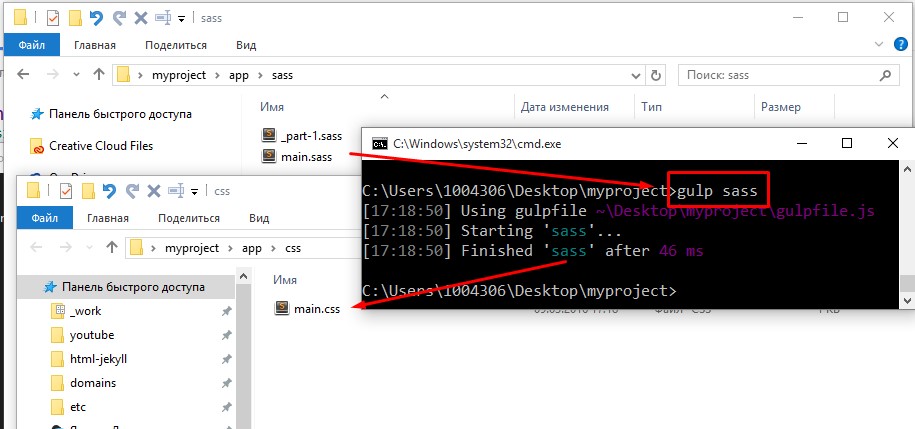


Рисунок 15-gulp sass

### Наблюдение за изменениями в файлах (Gulp Watch)

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

|  |
| --- |
| gulp.watch('watch-files', ['task1', 'task2']); |

Если, например, решить наблюдать за всеми изменениями в файлах sass проекта, то можно использовать следующую конструкцию:

|  |
| --- |
| gulp.watch('app/sass/\*\*/\*.sass', ['sass']); |

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

Также, можем создать отдельный таск для наблюдения за всеми необходимыми файлами

|  |  |
| --- | --- |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', ['sass']); *// Наблюдение за sass файлами* |
|  | *// Наблюдение за другими типами файлов* |
|  | }); |

Для Gulp 4 код будет выглядеть так:

|  |  |
| --- | --- |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); |
|  | }); |

Если запустить 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](https://www.browsersync.io/).

### Автоматическое обновление страниц с использованием Bbrowser 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:

|  |
| --- |
| npm i browser-sync --save-dev |

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

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'); *// Подключаем Browser Sync* |

Создаем таск для Browser Sync:

|  |  |
| --- | --- |
|  | gulp.task('browser-sync', function() { *// Создаем таск browser-sync* |
|  | browserSync({ *// Выполняем browser Sync* |
|  | server: { *// Определяем параметры сервера* |
|  | baseDir: 'app' *// Директория для сервера - app* |
|  | }, |
|  | notify: false *// Отключаем уведомления* |
|  | }); |
|  | }); |

Отлично! Сервер для работы и автоматической перезагрузки готов. Теперь давайте последим за изменениями в Sass. Если файл Sass обновляется, автоматически вставляем в HTML измененный CSS файл:

|  |  |
| --- | --- |
|  | gulp.task('sass', function(){ *// Создаем таск Sass* |
|  | return gulp.src('app/sass/\*\*/\*.sass') *// Берем источник* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(gulp.dest('app/css')) *// Выгружаем результата в папку app/css* |
|  | .pipe(browserSync.reload({stream: true})) *// Обновляем CSS на странице при изменении* |
|  | }); |

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

|  |  |
| --- | --- |
|  | gulp.task('watch', ['sass', 'browser-sync'], function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', ['sass']); *// Наблюдение за sass файлами* |
|  | *// Наблюдение за другими типами файлов* |
|  | }); |

Выполняем таски ['sass', 'browser-sync'] до запуска **watch**, так как их выполнение необходимо нам для корректного отображения изменений на момент запуска сервера.

Для Gulp 4 логичнее написать так и выполнять всю конструкцию в стандартном таске:

|  |  |
| --- | --- |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); |
|  | }); |
|  | gulp.task('default', gulp.parallel('sass', 'browser-sync', 'watch')); |

Расположим таск watch после всех других тасков и в результате получим такой **gulpfile.js** для Gulp 3:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'); *// Подключаем Browser Sync* |
|  |  |
|  | gulp.task('sass', function(){ *// Создаем таск Sass* |
|  | return gulp.src('app/sass/\*\*/\*.sass') *// Берем источник* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(gulp.dest('app/css')) *// Выгружаем результата в папку app/css* |
|  | .pipe(browserSync.reload({stream: true})) *// Обновляем CSS на странице при изменении* |
|  | }); |
|  |  |
|  | gulp.task('browser-sync', function() { *// Создаем таск browser-sync* |
|  | browserSync({ *// Выполняем browserSync* |
|  | server: { *// Определяем параметры сервера* |
|  | baseDir: 'app' *// Директория для сервера - app* |
|  | }, |
|  | notify: false *// Отключаем уведомления* |
|  | }); |
|  | }); |
|  |  |
|  | gulp.task('watch', ['sass', 'browser-sync'], function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', ['sass']); *// Наблюдение за sass файлами* |
|  | *// Наблюдение за другими типами файлов* |
|  | }); |

Такой код получится для Gulp 4:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'); *// Подключаем Browser Sync* |
|  |  |
|  | gulp.task('sass', function(){ *// Создаем таск Sass* |
|  | return gulp.src('app/sass/\*\*/\*.sass') *// Берем источник* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(gulp.dest('app/css')) *// Выгружаем результата в папку app/css* |
|  | .pipe(browserSync.reload({stream: true})) *// Обновляем CSS на странице при изменении* |
|  | }); |
|  |  |
|  | gulp.task('browser-sync', function() { *// Создаем таск browser-sync* |
|  | browserSync({ *// Выполняем browserSync* |
|  | server: { *// Определяем параметры сервера* |
|  | baseDir: 'app' *// Директория для сервера - app* |
|  | }, |
|  | notify: false *// Отключаем уведомления* |
|  | }); |
|  | }); |
|  |  |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); *// Наблюдение за sass файлами* |
|  | }); |
|  | gulp.task('default', gulp.parallel('sass', 'browser-sync', 'watch')); |
|  |  |

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

|  |  |
| --- | --- |
|  | <!DOCTYPE html> |
|  | <html lang="en"> |
|  | <head> |
|  | <meta charset="UTF-8"> |
|  | <title>Document</title> |
|  | <link rel="stylesheet" href="css/main.css"> |
|  | </head> |
|  | <body> |
|  |  |
|  | </body> |
|  | </html> |

Выполним в терминале команду "gulp". Результат представлен на рисунке 16.

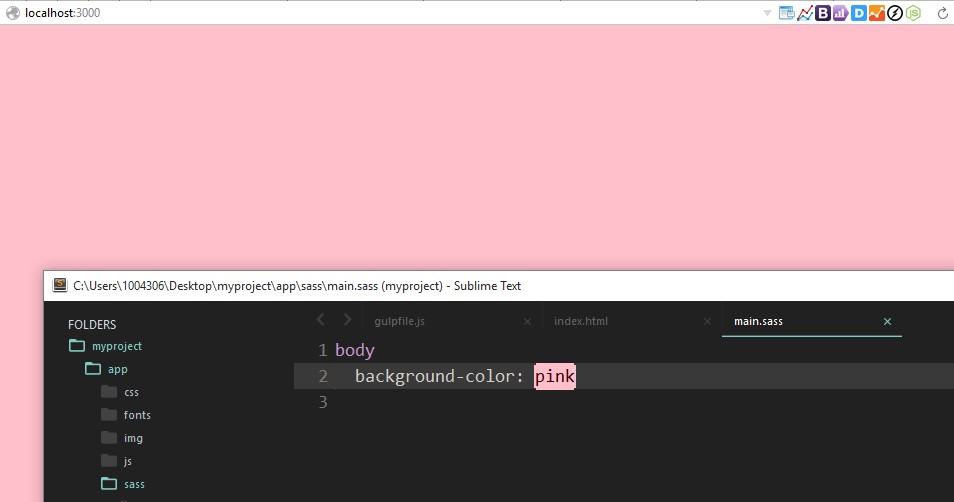


Рисунок 16-main.sass

Разберемся, что у нас происходит в консоли (картина может разниться, в зависимости от версии ПО). На рисунке 17 cmd.exe.

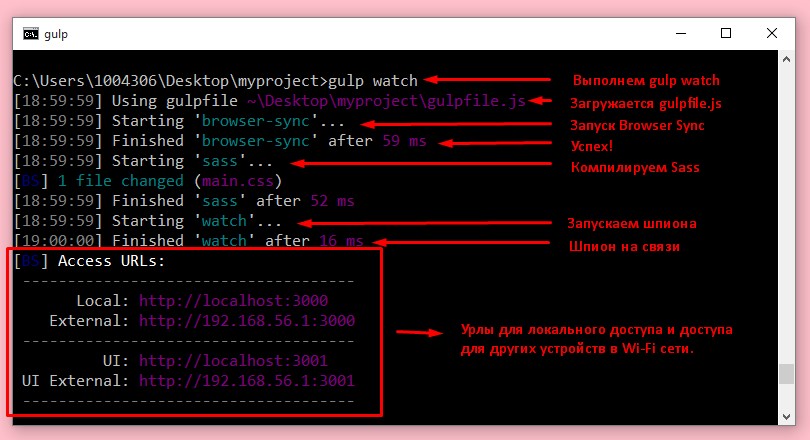


Рисунок 17-cmd.exe

Для обновления страницы при сохранении HTML и JS. Создайте в папке app/js файл **common.js**. Это основной пользовательский JS файл в нашем проекте. Модифицируем код:

Код для Gulp 3:

|  |  |
| --- | --- |
|  | gulp.task('watch', ['sass', 'browser-sync'], function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', ['sass']); *// Наблюдение за sass файлами в папке sass* |
|  | gulp.watch('app/\*.html', browserSync.reload); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], browserSync.reload); *// Наблюдение за главным JS файлом и за библиотеками* |
|  | }); |

Код для Gulp 4:

|  |  |
| --- | --- |
|  | gulp.task('scripts', function() { |
|  | return gulp.src(['app/js/common.js', 'app/libs/\*\*/\*.js']) |
|  | .pipe(browserSync.reload({ stream: true })) |
|  | }); |
|  |  |
|  | gulp.task('code', function() { |
|  | return gulp.src('app/\*.html') |
|  | .pipe(browserSync.reload({ stream: true })) |
|  | }); |
|  |  |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); *// Наблюдение за sass файлами* |
|  | gulp.watch('app/\*.html', gulp.parallel('code')); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], gulp.parallel('scripts')); *// Наблюдение за главным JS файлом и за библиотеками* |
|  | }); |
|  | gulp.task('default', gulp.parallel('sass', 'browser-sync', 'watch')); |

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

Уже имеем довольно продвинутое рабочее окружение. Но двигаемся дальше, это не все, на что способен Gulp.

### Оптимизация JavaScript

Рассмотрим, как можно оптимизировать JS файлы проекта. Чаще всего, в оптимизации нуждаются библиотеки и сторонние jQuery и JavaScript плагины. Создадим в папке **app** папку **libs**, которая будет содержать необходимые проекту библиотеки. Все библиотеки будем размещать в отдельных папках. Для установки новых библиотек используем [Bower](http://bower.io/).

Установим Bower:

|  |
| --- |
| npm i -g bower |

Для работы Bower необходим установленный [Git](http://git-scm.com/downloads). Если в ОС Windows используется оболочка Ubuntu bash, то установку Git выполнять не нужно.

Теперь в папке проекта создаем файл **.bowerrc**, в который напишем:

|  |  |
| --- | --- |
|  | { |
|  | "directory" : "app/libs/" |
|  | } |

Установим jQuery и Magnific Popup:

|  |
| --- |
| bower i jquery magnific-popup |

На рисунке 18 представлена структура папки libs.

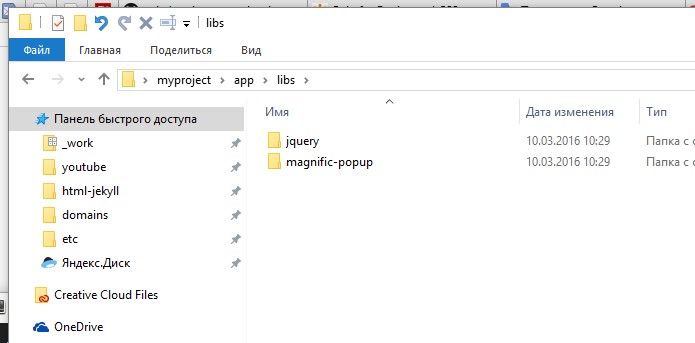


Рисунок 18-папка libs

Все (ну, или почти все) плагины имеют папку dist. В этой папке располагаются готовые файлы продакшена, которые и будем использовать в проекте.

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

|  |
| --- |
| npm i --save-dev gulp-concat gulp-uglifyjs |

Подключим новые библиотеки в gulpfile.js:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'); *// Подключаем gulp-uglifyjs (для сжатия JS)* |

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

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

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

|  |
| --- |
| gulp scripts |

На рисунке 19 представлена cmd.exe с запросом gulp scripts.

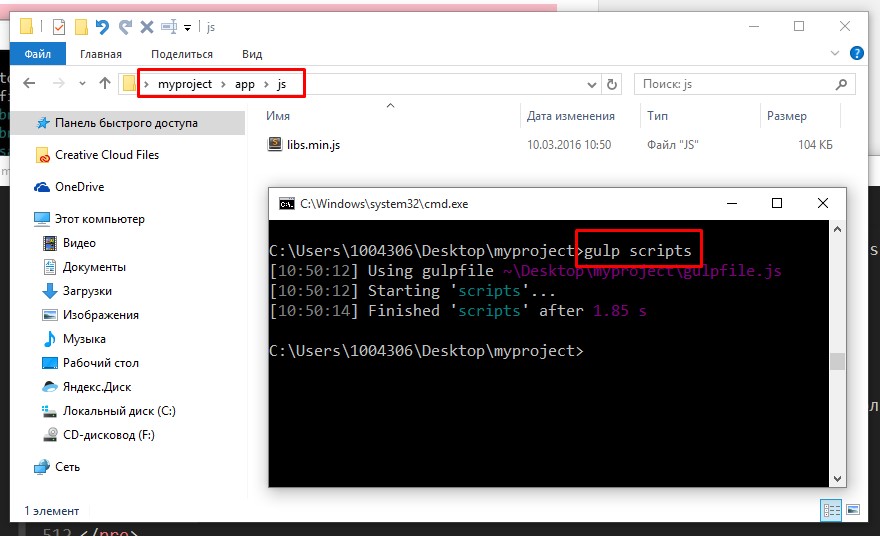
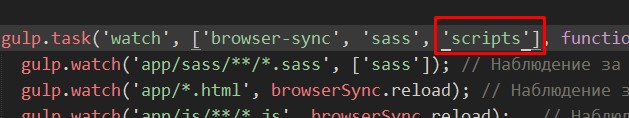


Рисунок 19- запрос gulp scripts

Выполнение таска **scripts** можно запустить перед выполнением watch. Gulp 3:



Для Gulp 4 код будет выглядеть следующим образом - добавим в параллельное выполнение таска **scripts** (некоторую структуру уже задали ранее):

|  |  |
| --- | --- |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); *// Наблюдение за sass файлами* |
|  | gulp.watch('app/\*.html', gulp.parallel('code')); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], gulp.parallel('scripts')); *// Наблюдение за главным JS файлом и за библиотеками* |
|  | }); |
|  | gulp.task('default', gulp.parallel('sass', 'scripts', 'browser-sync', 'watch')); |

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 фале **sass/libs.sass**:

|  |
| --- |
| @import "app/libs/magnific-popup/dist/magnific-popup.css" *// Импортируем библиотеку Magnific Popup* |

В новых версиях **gulp-sass** для импорта CSS файлов в Sass необходимо указывать расширение .css и импортировать CSS файлы в SASS файлы с нижним подчёркиванием в начале названия. Например, для того, чтобы импортировать файл library-name.css, необходимо создать вспомогатальный SASS файл, например, \_libs.sass, импортировать в него нужный CSS - **@import "app/libs/library-name.css"** и добавить вспомогательный \_libs.sass в главный main.sass без указания нижнего подчёркивания и расширения, например, так: **@import "libs"**

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

Если в файле libs.css не появляется кода библиотек, а вы по-прежнему видите в нём конструкции @import, создайте отдельный файл \_libs.sass для библиотек, который начинался бы**с нижнего подчёркивания**. Затем импортируйте этот файл в главный, тем самым объединив и библиотеки, и пользовательские стили в один файл.

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

|  |
| --- |
| npm i gulp-cssnano gulp-rename *--save-dev* |

И подключим их в нашем gulpfile.js:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'); *// Подключаем библиотеку для переименования файлов* |

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

|  |  |
| --- | --- |
|  | gulp.task('css-libs', ['sass'], function() { |
|  | return gulp.src('app/sass/libs.sass') *// Выбираем файл для минификации* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(cssnano()) *// Сжимаем* |
|  | .pipe(rename({suffix: '.min'})) *// Добавляем суффикс .min* |
|  | .pipe(gulp.dest('app/css')); *// Выгружаем в папку app/css* |
|  | }); |
|  |  |
|  | gulp.task('watch', ['browser-sync', 'css-libs', 'scripts'], function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', ['sass']); *// Наблюдение за sass файлами в папке sass* |
|  | gulp.watch('app/\*.html', browserSync.reload); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], browserSync.reload); *// Наблюдение за главным JS файлом и за библиотеками* |
|  | }); |

Код для Gulp 4:

|  |  |
| --- | --- |
|  | gulp.task('css-libs', function() { |
|  | return gulp.src('app/sass/libs.sass') *// Выбираем файл для минификации* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(cssnano()) *// Сжимаем* |
|  | .pipe(rename({suffix: '.min'})) *// Добавляем суффикс .min* |
|  | .pipe(gulp.dest('app/css')); *// Выгружаем в папку app/css* |
|  | }); |
|  |  |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); *// Наблюдение за sass файлами* |
|  | gulp.watch('app/\*.html', gulp.parallel('code')); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], gulp.parallel('scripts')); *// Наблюдение за главным JS файлом и за библиотеками* |
|  | }); |
|  | gulp.task('default', gulp.parallel('css-libs', 'sass', 'scripts', 'browser-sync', 'watch')); |

### Подготовка к продакшену

Результирующий код для **Gulp 4** будет представлен в конце статьи.

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

|  |  |
| --- | --- |
|  | gulp.task('build', ['sass', 'scripts'], function() { |
|  |  |
|  | var buildCss = gulp.src([ *// Переносим CSS стили в продакшен* |
|  | 'app/css/main.css', |
|  | 'app/css/libs.min.css' |
|  | ]) |
|  | .pipe(gulp.dest('dist/css')) |
|  |  |
|  | var buildFonts = gulp.src('app/fonts/\*\*/\*') *// Переносим шрифты в продакшен* |
|  | .pipe(gulp.dest('dist/fonts')) |
|  |  |
|  | var buildJs = gulp.src('app/js/\*\*/\*') *// Переносим скрипты в продакшен* |
|  | .pipe(gulp.dest('dist/js')) |
|  |  |
|  | var buildHtml = gulp.src('app/\*.html') *// Переносим HTML в продакшен* |
|  | .pipe(gulp.dest('dist')); |
|  |  |
|  | }); |

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

Перед тем, как собирать проект желательно очистить папку dist, чтобы не оставалось лишних записей от предыдущих итераций с нашим проектом.

Установим и подключим пакет del:

|  |
| --- |
| npm i del --save-dev |

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'), *// Подключаем библиотеку для переименования файлов* |
|  | del = require('del'); *// Подключаем библиотеку для удаления файлов и папок* |

Создаем таск очистки **clean** и добавляем его выполнение перед выполнение build:

|  |  |
| --- | --- |
|  | gulp.task('clean', function() { |
|  | return del.sync('dist'); *// Удаляем папку dist перед сборкой* |
|  | }); |
|  |  |
|  | gulp.task('build', ['clean', 'sass', 'scripts'], function() { |
|  |  |
|  | var buildCss = gulp.src([ *// Переносим библиотеки в продакшен* |
|  | 'app/css/main.css', |
|  | 'app/css/libs.min.css' |
|  | ]) |
|  | .pipe(gulp.dest('dist/css')) |
|  |  |
|  | var buildFonts = gulp.src('app/fonts/\*\*/\*') *// Переносим шрифты в продакшен* |
|  | .pipe(gulp.dest('dist/fonts')) |
|  |  |
|  | var buildJs = gulp.src('app/js/\*\*/\*') *// Переносим скрипты в продакшен* |
|  | .pipe(gulp.dest('dist/js')) |
|  |  |
|  | var buildHtml = gulp.src('app/\*.html') *// Переносим HTML в продакшен* |
|  | .pipe(gulp.dest('dist')); |
|  |  |
|  | }); |

Для Gulp 4 попробуйте составить таски самостоятельно, как мы это делали в предыдущих примерах.

### Оптимизация изображений

Как могли заметить, в проекте на продакшене не хватает изображений. Добавим обработку изображений в проект. Данный раздел выполнен с использованием Gulp 3. Код для Gulp 4 можно адаптировать самостоятельно, как мы это делали ранее.

В папке app/img есть 3 изображения, которые необходимо перенести в папку продакшена, оптимизируя. На рисунке 20 изображена папка app/img.

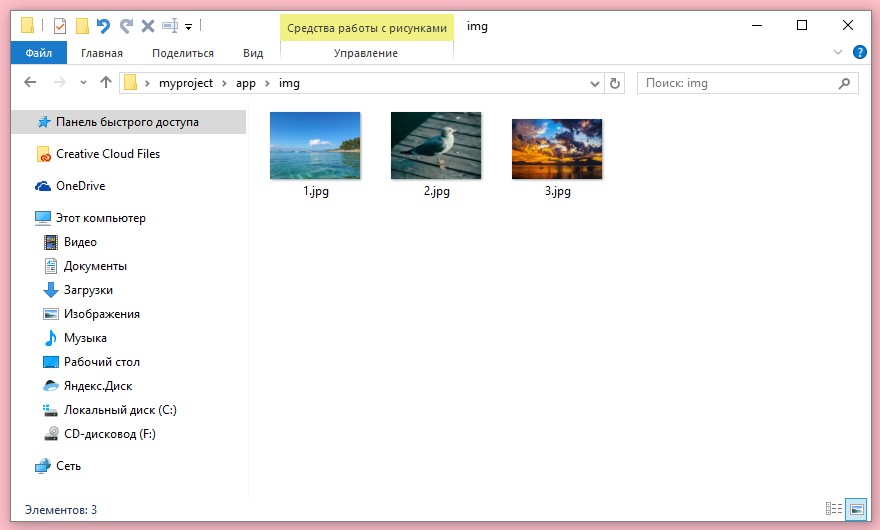


Рисунок 20-папка img

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

|  |
| --- |
| npm i gulp-imagemin imagemin-pngquant --save-dev |

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'), *// Подключаем библиотеку для переименования файлов* |
|  | del = require('del'), *// Подключаем библиотеку для удаления файлов и папок* |
|  | imagemin = require('gulp-imagemin'), *// Подключаем библиотеку для работы с изображениями* |
|  | pngquant = require('imagemin-pngquant'); *// Подключаем библиотеку для работы с png* |

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

|  |  |
| --- | --- |
|  | gulp.task('img', function() { |
|  | return gulp.src('app/img/\*\*/\*') *// Берем все изображения из app* |
|  | .pipe(imagemin({ *// Сжимаем их с наилучшими настройками* |
|  | interlaced: true, |
|  | progressive: true, |
|  | svgoPlugins: [{removeViewBox: false}], |
|  | use: [pngquant()] |
|  | })) |
|  | .pipe(gulp.dest('dist/img')); *// Выгружаем на продакшен* |
|  | }); |
|  |  |
|  | gulp.task('build', ['clean', 'img', 'sass', 'scripts'], function() { |
|  |  |
|  | var buildCss = gulp.src([ *// Переносим библиотеки в продакшен* |
|  | 'app/css/main.css', |
|  | 'app/css/libs.min.css' |
|  | ]) |
|  | .pipe(gulp.dest('dist/css')) |
|  |  |
|  | var buildFonts = gulp.src('app/fonts/\*\*/\*') *// Переносим шрифты в продакшен* |
|  | .pipe(gulp.dest('dist/fonts')) |
|  |  |
|  | var buildJs = gulp.src('app/js/\*\*/\*') *// Переносим скрипты в продакшен* |
|  | .pipe(gulp.dest('dist/js')) |
|  |  |
|  | var buildHtml = gulp.src('app/\*.html') *// Переносим HTML в продакшен* |
|  | .pipe(gulp.dest('dist')); |
|  |  |
|  | }); |

Большое количество картинок будет обрабатываться значительно дольше, поэтому к обработке изображений надо добавить кеш, чтобы картинки кешировались, экономя время.

Установим подключим **gulp-cache**:

|  |
| --- |
| npm i gulp-cache *--save-dev* |

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'), *// Подключаем библиотеку для переименования файлов* |
|  | del = require('del'), *// Подключаем библиотеку для удаления файлов и папок* |
|  | imagemin = require('gulp-imagemin'), *// Подключаем библиотеку для работы с изображениями* |
|  | pngquant = require('imagemin-pngquant'), *// Подключаем библиотеку для работы с png* |
|  | cache = require('gulp-cache'); *// Подключаем библиотеку кеширования* |

Модифицируем таск **img**:

|  |  |
| --- | --- |
|  | gulp.task('img', function() { |
|  | return gulp.src('app/img/\*\*/\*') *// Берем все изображения из app* |
|  | .pipe(cache(imagemin({ *// Сжимаем их с наилучшими настройками с учетом кеширования* |
|  | interlaced: true, |
|  | progressive: true, |
|  | svgoPlugins: [{removeViewBox: false}], |
|  | use: [pngquant()] |
|  | }))) |
|  | .pipe(gulp.dest('dist/img')); *// Выгружаем на продакшен* |
|  | }); |

### Автоматическое создание префиксов CSS с помощью Gulp

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

|  |
| --- |
| display: flex |

Мы получили на выходе:

|  |  |
| --- | --- |
|  | display: -webkit-flex; |
|  | display: -moz-flex; |
|  | display: -ms-flex; |
|  | display: -o-flex; |
|  | display: flex; |

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

|  |
| --- |
| npm i --save-dev gulp-autoprefixer |

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'), *// Подключаем библиотеку для переименования файлов* |
|  | del = require('del'), *// Подключаем библиотеку для удаления файлов и папок* |
|  | imagemin = require('gulp-imagemin'), *// Подключаем библиотеку для работы с изображениями* |
|  | pngquant = require('imagemin-pngquant'), *// Подключаем библиотеку для работы с png* |
|  | cache = require('gulp-cache'), *// Подключаем библиотеку кеширования* |
|  | autoprefixer = require('gulp-autoprefixer');*// Подключаем библиотеку для автоматического добавления префиксов* |

И модифицируем наш таск **sass**:

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

### Стандартный таск Gulp

Итак, имеем 2 главных таска - **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 таск **watch**, можно повесить его на дефолтный таск, чтобы не писать в консоли постоянно gulp watch, а писать просто **gulp**.

|  |
| --- |
| gulp.task('default', ['watch']); |

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

|  |  |
| --- | --- |
|  | gulp.task('clear', function () { |
|  | return cache.clearAll(); |
|  | }) |

Если возникнут проблемы с изображениями или другими кешируемыми файлами, надо просто почистить кеш.

В результате, должен получиться такой **gulpfile.js**. Gulp 3:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'), *// Подключаем библиотеку для переименования файлов* |
|  | del = require('del'), *// Подключаем библиотеку для удаления файлов и папок* |
|  | imagemin = require('gulp-imagemin'), *// Подключаем библиотеку для работы с изображениями* |
|  | pngquant = require('imagemin-pngquant'), *// Подключаем библиотеку для работы с png* |
|  | cache = require('gulp-cache'), *// Подключаем библиотеку кеширования* |
|  | autoprefixer = require('gulp-autoprefixer');*// Подключаем библиотеку для автоматического добавления префиксов* |
|  |  |
|  | gulp.task('sass', function(){ *// Создаем таск Sass* |
|  | return gulp.src('app/sass/\*\*/\*.sass') *// Берем источник* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], { cascade: true })) *// Создаем префиксы* |
|  | .pipe(gulp.dest('app/css')) *// Выгружаем результата в папку app/css* |
|  | .pipe(browserSync.reload({stream: true})) *// Обновляем CSS на странице при изменении* |
|  | }); |
|  |  |
|  | gulp.task('browser-sync', function() { *// Создаем таск browser-sync* |
|  | browserSync({ *// Выполняем browserSync* |
|  | server: { *// Определяем параметры сервера* |
|  | baseDir: 'app' *// Директория для сервера - app* |
|  | }, |
|  | notify: false *// Отключаем уведомления* |
|  | }); |
|  | }); |
|  |  |
|  | gulp.task('scripts', function() { |
|  | return gulp.src([ *// Берем все необходимые библиотеки* |
|  | 'app/libs/jquery/dist/jquery.min.js', *// Берем jQuery* |
|  | 'app/libs/magnific-popup/dist/jquery.magnific-popup.min.js' *// Берем Magnific Popup* |
|  | ]) |
|  | .pipe(concat('libs.min.js')) *// Собираем их в кучу в новом файле libs.min.js* |
|  | .pipe(uglify()) *// Сжимаем JS файл* |
|  | .pipe(gulp.dest('app/js')); *// Выгружаем в папку app/js* |
|  | }); |
|  |  |
|  | gulp.task('css-libs', ['sass'], function() { |
|  | return gulp.src('app/css/libs.sass') *// Выбираем файл для минификации* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(cssnano()) *// Сжимаем* |
|  | .pipe(rename({suffix: '.min'})) *// Добавляем суффикс .min* |
|  | .pipe(gulp.dest('app/css')); *// Выгружаем в папку app/css* |
|  | }); |
|  |  |
|  | gulp.task('clean', function() { |
|  | return del.sync('dist'); *// Удаляем папку dist перед сборкой* |
|  | }); |
|  |  |
|  | gulp.task('img', function() { |
|  | return gulp.src('app/img/\*\*/\*') *// Берем все изображения из app* |
|  | .pipe(cache(imagemin({ *// С кешированием* |
|  | *// .pipe(imagemin({ // Сжимаем изображения без кеширования* |
|  | interlaced: true, |
|  | progressive: true, |
|  | svgoPlugins: [{removeViewBox: false}], |
|  | use: [pngquant()] |
|  | }))/\*\*/) |
|  | .pipe(gulp.dest('dist/img')); *// Выгружаем на продакшен* |
|  | }); |
|  |  |
|  | gulp.task('build', ['clean', 'img', 'sass', 'scripts'], function() { |
|  |  |
|  | var buildCss = gulp.src([ *// Переносим библиотеки в продакшен* |
|  | 'app/css/main.css', |
|  | 'app/css/libs.min.css' |
|  | ]) |
|  | .pipe(gulp.dest('dist/css')) |
|  |  |
|  | var buildFonts = gulp.src('app/fonts/\*\*/\*') *// Переносим шрифты в продакшен* |
|  | .pipe(gulp.dest('dist/fonts')) |
|  |  |
|  | var buildJs = gulp.src('app/js/\*\*/\*') *// Переносим скрипты в продакшен* |
|  | .pipe(gulp.dest('dist/js')) |
|  |  |
|  | var buildHtml = gulp.src('app/\*.html') *// Переносим HTML в продакшен* |
|  | .pipe(gulp.dest('dist')); |
|  |  |
|  | }); |
|  |  |
|  | gulp.task('clear', function (callback) { |
|  | return cache.clearAll(); |
|  | }); |
|  |  |
|  | gulp.task('watch', ['browser-sync', 'css-libs', 'scripts'], function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', ['sass']); *// Наблюдение за sass файлами в папке sass* |
|  | gulp.watch('app/\*.html', browserSync.reload); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], browserSync.reload); *// Наблюдение за JS файлами в папке js* |
|  | }); |
|  |  |
|  | gulp.task('default', ['watch']); |

Результирующий код для Gulp 4:

|  |  |
| --- | --- |
|  | var gulp = require('gulp'), *// Подключаем Gulp* |
|  | sass = require('gulp-sass'), *//Подключаем Sass пакет,* |
|  | browserSync = require('browser-sync'), *// Подключаем Browser Sync* |
|  | concat = require('gulp-concat'), *// Подключаем gulp-concat (для конкатенации файлов)* |
|  | uglify = require('gulp-uglifyjs'), *// Подключаем gulp-uglifyjs (для сжатия JS)* |
|  | cssnano = require('gulp-cssnano'), *// Подключаем пакет для минификации CSS* |
|  | rename = require('gulp-rename'), *// Подключаем библиотеку для переименования файлов* |
|  | del = require('del'), *// Подключаем библиотеку для удаления файлов и папок* |
|  | imagemin = require('gulp-imagemin'), *// Подключаем библиотеку для работы с изображениями* |
|  | pngquant = require('imagemin-pngquant'), *// Подключаем библиотеку для работы с png* |
|  | cache = require('gulp-cache'), *// Подключаем библиотеку кеширования* |
|  | autoprefixer = require('gulp-autoprefixer');*// Подключаем библиотеку для автоматического добавления префиксов* |
|  |  |
|  | gulp.task('sass', function() { *// Создаем таск Sass* |
|  | return gulp.src('app/sass/\*\*/\*.sass') *// Берем источник* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], { cascade: true })) *// Создаем префиксы* |
|  | .pipe(gulp.dest('app/css')) *// Выгружаем результата в папку app/css* |
|  | .pipe(browserSync.reload({stream: true})) *// Обновляем CSS на странице при изменении* |
|  | }); |
|  |  |
|  | gulp.task('browser-sync', function() { *// Создаем таск browser-sync* |
|  | browserSync({ *// Выполняем browserSync* |
|  | server: { *// Определяем параметры сервера* |
|  | baseDir: 'app' *// Директория для сервера - app* |
|  | }, |
|  | notify: false *// Отключаем уведомления* |
|  | }); |
|  | }); |
|  |  |
|  | gulp.task('scripts', function() { |
|  | return gulp.src([ *// Берем все необходимые библиотеки* |
|  | 'app/libs/jquery/dist/jquery.min.js', *// Берем jQuery* |
|  | 'app/libs/magnific-popup/dist/jquery.magnific-popup.min.js' *// Берем Magnific Popup* |
|  | ]) |
|  | .pipe(concat('libs.min.js')) *// Собираем их в кучу в новом файле libs.min.js* |
|  | .pipe(uglify()) *// Сжимаем JS файл* |
|  | .pipe(gulp.dest('app/js')); *// Выгружаем в папку app/js* |
|  | }); |
|  |  |
|  | gulp.task('code', function() { |
|  | return gulp.src('app/\*.html') |
|  | .pipe(browserSync.reload({ stream: true })) |
|  | }); |
|  |  |
|  | gulp.task('css-libs', function() { |
|  | return gulp.src('app/css/libs.sass') *// Выбираем файл для минификации* |
|  | .pipe(sass()) *// Преобразуем Sass в CSS посредством gulp-sass* |
|  | .pipe(cssnano()) *// Сжимаем* |
|  | .pipe(rename({suffix: '.min'})) *// Добавляем суффикс .min* |
|  | .pipe(gulp.dest('app/css')); *// Выгружаем в папку app/css* |
|  | }); |
|  |  |
|  | gulp.task('clean', async function() { |
|  | return del.sync('dist'); *// Удаляем папку dist перед сборкой* |
|  | }); |
|  |  |
|  | gulp.task('img', function() { |
|  | return gulp.src('app/img/\*\*/\*') *// Берем все изображения из app* |
|  | .pipe(cache(imagemin({ *// С кешированием* |
|  | *// .pipe(imagemin({ // Сжимаем изображения без кеширования* |
|  | interlaced: true, |
|  | progressive: true, |
|  | svgoPlugins: [{removeViewBox: false}], |
|  | use: [pngquant()] |
|  | }))/\*\*/) |
|  | .pipe(gulp.dest('dist/img')); *// Выгружаем на продакшен* |
|  | }); |
|  |  |
|  | gulp.task('prebuild', async function() { |
|  |  |
|  | var buildCss = gulp.src([ *// Переносим библиотеки в продакшен* |
|  | 'app/css/main.css', |
|  | 'app/css/libs.min.css' |
|  | ]) |
|  | .pipe(gulp.dest('dist/css')) |
|  |  |
|  | var buildFonts = gulp.src('app/fonts/\*\*/\*') *// Переносим шрифты в продакшен* |
|  | .pipe(gulp.dest('dist/fonts')) |
|  |  |
|  | var buildJs = gulp.src('app/js/\*\*/\*') *// Переносим скрипты в продакшен* |
|  | .pipe(gulp.dest('dist/js')) |
|  |  |
|  | var buildHtml = gulp.src('app/\*.html') *// Переносим HTML в продакшен* |
|  | .pipe(gulp.dest('dist')); |
|  |  |
|  | }); |
|  |  |
|  | gulp.task('clear', function (callback) { |
|  | return cache.clearAll(); |
|  | }) |
|  |  |
|  | gulp.task('watch', function() { |
|  | gulp.watch('app/sass/\*\*/\*.sass', gulp.parallel('sass')); *// Наблюдение за sass файлами* |
|  | gulp.watch('app/\*.html', gulp.parallel('code')); *// Наблюдение за HTML файлами в корне проекта* |
|  | gulp.watch(['app/js/common.js', 'app/libs/\*\*/\*.js'], gulp.parallel('scripts')); *// Наблюдение за главным JS файлом и за библиотеками* |
|  | }); |
|  | gulp.task('default', gulp.parallel('css-libs', 'sass', 'scripts', 'browser-sync', 'watch')); |
|  | gulp.task('build', gulp.parallel('prebuild', 'clean', 'img', 'sass', 'scripts')); |

## 3 СИСТЕМА ОТСЛЕЖИВАНИЯ ОШИБОК

При написании плагина к gulp рекомендуется испускать событие error при возникновении ошибок во время работы плагина. Но nodejs-потоки, на которых основана система сборки, не позволяют ошибкам оставаться незамеченными. В случае, если на событие error никто не подписался, выбросится исключение, чтобы сообщение точно достигло пользователя. В результате, при работе с gulp разработчики часто видят такие ошибки

events.js:72  
throw er; // Unhandled 'error' event

При применении Continious-Integration (CI), когда на каждый коммит запускаются автоматические проверки, это может быть и полезным (сборка провалилась, билд не собрался, ответственные получат письма). Но вечно падающий watcher в локальной разработке, который нужно постоянно перезапускать – это большая неприятность.

Решение:

Можно подписаться на событие error:

gulp.task('less', function() {  
 return gulp.src('less/\*.less')  
 .pipe(less().on('error', gutil.log))  
 .pipe(gulp.dest('app/css'));  
});

Можно подключить плагин gulp-plumber, который не только подпишется на error для одного плагина, но и автоматически сделает это и для всех последующих плагинов, подключенных через pipe:

gulp.task('less', function() {  
 return gulp.src('less/\*.less')  
 .pipe(plumber())  
 .pipe(less())  
 .pipe(gulp.dest('app/css'));  
});

Допустим у нас есть CI-сервер, где идет сборка наших скриптов для выкладки. Чтобы наша сборка работала вместе с watch, мы применили одно из решений выше. Но теперь оказывается, что при ошибках в сборке наш билд все равно отмечается как успешный. Команда gulp всегда завершается с кодом 0. Получается, для CI-сборки нам не нужно проглатывать ошибки. Можно добавлять свой обработчик ошибок только для watch режима, но это усложнит описание сборки и повысит шансы допустить ошибку. Есть решение, как настроить сборку одинаково, но при этом поддержать работу и в режиме билда и в режиме watch.

На самом деле gulp пытается слушать ошибки в потоках файлов. Но из-за того, что обычно последним идет вызов gulp.dest(), записывающий результат на диск, а ошибки случаются в промежуточных плагинах, то ошибка не достигает конца цепочки, поэтому gulp о ней не узнает. Например рассмотрим такую цепочку:

stream1.on('error', function() { ***console***.log('error 1') })  
stream2.on('error', function() { ***console***.log('error 2') })  
stream3.on('error', function() { ***console***.log('error 3') })  
stream1  
 .pipe(stream2)  
 .pipe(stream3);  
stream1.emit('error');  
// в консоли выведется только "error 1"

В отличие от, например, promise, ошибки в потоках не распространяются дальше по цепочке, их нужно перехватывать в каждом потоке отдельно. По этому поводу есть pull-request в io.js, но в нынешних версиях передать ошибку в конец цепочки не получится. Поэтому gulp не может перехватить ошибки промежуточных потоках и нам это нужно делать самостоятельно.

Зато gulp в качестве описания task принимает не только функцию, возвращающую поток, но и обычную callback-style функцию, как и многие API в node.js. В такой функции будем решать, когда задача завершилось с ошибкой, а когда успешно:

gulp.task('less', function(done) {  
 gulp.src('less/\*.less')  
 .pipe(less().on('error', function(error) {  
 // у нас ошибка  
 done(error);  
 }))  
 .pipe(gulp.dest('app/css'))  
 .on('end', function() {  
 // у нас все закончилось успешно  
 done();  
 });  
});

Такое описание сборки выглядит немного больше, потому что мы теперь делаем часть работы за gulp, но теперь мы ее делаем правильно. А если написать себе вспомогательную функцию,  вроде такой, то разница будет практически незаметна:

gulp.task('less', wrapPipe(function(success, error) {  
 return gulp.src('less/\*.less')  
 .pipe(less().on('error', error))  
 .pipe(gulp.dest('app/css'));  
}));

Зато получим правильные сообщения в консоли как во время разработки и пересборки при сохранении, так и на CI-сервере во время билда.

# ЗАКЛЮЧЕНИЕ

Мы рассмотрели большинство основных способов настройки клиента и сервера Git с тем, чтобы он был максимально соответствовал вашим рабочим процессам и проектам.

Мы узнали о всевозможных настройках, атрибутах файлов и о перехватчиках событий, а также рассмотрели пример настройки сервера с соблюдением политики. Теперь вам должно быть по плечу заставить Git подстроиться под практически любой тип рабочего процесса.

Также узнали о Gulpfile: инструмент для автоматизации рутинных задач, которые возникают при веб-разработке. Это может быть не только frontend разработка, это может быть и backend разработка.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. .Git и Githab: что это такое и чем отличается [Электронный ресурс] **—**  https://tproger.ru/translations/difference-between-git-and-github/- статья в интернете.
2. Кузнецов Максим, Симдянов Игорь Объектно-ориентированное программирование на PHP[Письменный ресурс] **—** Спб.: «БХВ-Петербург», 2007. - С. 608. –Книга
3. PhpMyAdmin. [Электронный ресурс] **—**  https://ipipe.ru/info/phpmyadmin - статья в интернете.
4. Дамашке Гизберт PHP и MySQL [Письменный ресурс] **—** НТ Пресс - Москва, 2012. - 320 c.-книга
5. Gulpfile [Электронный ресурс] **—** https://webkyrs.info/page/chto-soboi-predstavliaet-gulpfile-js-i-zapusk-pervoi-zadachi – статья в интернете