Инициализация проекта

Проверяем наличие ноды

```
node --version
```

Проверяем наличие npm (Node Package Manager – дефолтный пакетный менеджер для JavaScript)

```
npm --version
```

Проверяем наличие npx

```
npx --version
```

Устанавливаем gulp-cli (устанавливаем пакет как глобальный)

```
npm install --global gulp-cli
```

Создадим директорию проекта и перейдем в неё

```
npx mkdirp test_project && cd my-project
```

Инициализируем проект, создав файл package.json, в котором будут отражены наши зависимости, названия проекта, ссылка на репозитория и т.д. и т.п.

```
npm init
```

Установим gulp в наш проект и добавим его в зависимость devDependencies в файле package.json с помощью флага --save-dev

```
npm install --save-dev gulp
```

Проверим версию gulp

```
gulp --version
```

Работа с проектом с помощью GULP

Как было сказано ранее, gulp конфигурируется с помощью файла, который называется gulpfile.js (или с большой буквы Gulpfile.js), поэтому давайте создадим его в корне нашего проекта.

```
touch gulpfile.js
```

Здесь стоит отметить, что gulpfile в больших проектах может сильно разрастись и поэтому его необходимо будет разделить на несколько. Это тоже возможно, но сегодня не будет рассмотрено.

Давайте теперь напишем нашу первую таску.

```
const { series } = require('gulp');

// callback в данном случае функция, которая вызовется после выполнения основного тела задачи
function clean(callback) {
  callback();
}

function build(callback) {
  callback();
}

exports.build = build;
exports.default = series(clean, build);
```

Пройдем по ней глазами и разберемся чуть подробней что за магия тут творится. В первой строке мы импортируем функцию series из библиотеки gulp.

Далее у нас перечислены сами задачи, которые мы можем выполнить с помощью cli. Стоит отметить, что у нас есть две области видимости публичная и приватная. Что к чему относится можно посмотреть с помощью команды `gulp --tasks`
Задачи из публичной области можно запустить с помощью одной команды `gulp` , а в коде мы обычно присваиваем их переменной exports.default.

Задачи из приватной области выполняются с указанием имени задачи, то есть `gulp build` в нашем случае.

Внутри каждой задачи мы видим вызов функции callback, который используется для сигнализации о завершении выполнения задачи. Когда задача завершает свою работу, она вызывает переданный callback, что сообщает Gulp, что задача завершена, и можно перейти к следующей задаче в цепочке.

Теперь немного поговорим о базовых элементах компоновщика gulp. В последней строке вы видите вызов функции **series**. Эта функция отвечает за последовательное выполнение всех тасков.
Помимо этой функции мы будем использовать несколько других.

Винил — это объект метаданных, описывающий файл. Основными свойствами экземпляра Vinyl являются pathи contents— основные аспекты файла в вашей файловой системе. Виниловые объекты можно использовать для описания файлов из многих источников — в локальной файловой системе или в любом удаленном хранилище.

task - Определяет задачу в gulp. После этого к задаче можно будет получить доступ из командной строки и из API series(), parallel().

parallel - Задачи, определенные в parallel, выполняются параллельно, независимо друг от друга

src - Создает поток для чтения объектов Винил из файловой системы

dest - Создает поток для записи объектов Винил в файловую систему

watch - Позволяет наблюдать за файликами и запускать задачу при возникновении изменений. Задачи обрабатываются единообразно с остальной частью системы задач.

Со всеми ними мы познакомимся чуть позже, а пока давайте перейдем к примеру, который позволит нам понять gulp чуть лучше.

Второй пример

Основная фишка этого компоновщика скрывается в его плагинах, которые мы можем установить дополнительно. Давайте поставим плагин, который позволяет минифицировать наш html код:

```
npm install --save-dev gulp-minify-html
```

Минификация поможет нам с тем, чтобы чуть уменьшить объем файла и вебсервер при выкатке изменений в прод отдавал файлы чуть быстрее.

Теперь давайте создадим в корневой папке папку, которая будет называться src и создадим там файл index.html.

```
mkdir src && touch index.html
```

Внутри hmtl файла напишем следующий код

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="dist/styles/main.css">
  <title>Test Project</title>
</head>
<body>
  <h1>Привет, Мир!</h1>
</body>
</html>
```

И теперь напишем следующий код в gulpfile.js

```
const gulp = require("gulp");
const minifyHtml = require("gulp-minify-html");

function build(callback) {
    return gulp.src('./src/*.html').pipe(minifyHtml()).pipe(gulp.dest('./build'));
}

exports.default = gulp.series(build);
```

Что в нем происходит? Сначала мы также импортируем библиотеки, но теперь у нас добавилась та библиотека, что мы поставили ранее.
Далее мы также объявляем задачу как и в прошлом примере, но в этот раз не вызываем callback. В Gulp, когда вы возвращаете результат выполнения потока (stream) из задачи, это само по себе сигнализирует Gulp о завершении задачи. В этом случае, поскольку возвращает поток, Gulp понимает, что задача завершена, и не требуется явно вызывать callback().

Что происходит в самой функции? Сначала мы создаем поток данных, который читает все файлы с расширением html в папке src и с помощью pipe передает их в функцию из библиотеки gulp-minify-html minifyHtml, которая как раз минифицирует файлы. Далее pipe передает поток данных в функцию dest, которая собирает все файлы в папке build.
Отдельно стоит остановиться на функции pipe, которая очень похожа на pipe из Линукса, типа `cat file | grep test` . Здесь тоже сначала читается файл, который передается в команду grep и далее уже ищется необходимая информация.
Использование pipe в gulpfile аналогично. Информация просто последовательно передается, пока не закончатся вызовы функций.

Ну и в конце мы создаем серию задач в публичном пространстве, то есть задачи можно запустить с помощью команды gulp и посмотреть на результат.

Третий пример

По своей сути он мало чем будет отличаться от предыдущего, только в этот раз мы поработаем с SASS.

Давайте также предварительно установим библиотеку:

```
npm install --save-dev gulp-sass
```

В папке src создадим еще один каталог styles и в нем файл main.scss.

```
mkdir styles && touch styles/main.scss
```

В main.scss добавим следующий код

```
body {
    font-family: Arial, sans-serif;
}

h1 {
    color: red;
}
```

В gulpfile добавим следующий код

```
const gulp = require('gulp');
const sass = require('gulp-sass')(require('sass'));

gulp.task('compile-sass', function() {
  return gulp.src('src/styles/*.scss')
    .pipe(sass())
    .pipe(gulp.dest('dist/styles'));
});

gulp.task('watch', function() {
  gulp.watch('src/styles/*.scss', gulp.series('compile-sass'));
});
```

В данном примере мы создали две задачи: одна просто компилирует sass код в css, а вторая будет наблюдать за изменением файлов в директории и если изменения получены, то будет запускаться первая задача на компиляцию.
Здесь мы использовали немного другой подход к созданию задач. Теперь мы это делаем через вызов функции task, куда первым параметром передаем название таски, а вторым анонимную функцию. Внутри таски compile-sass отличий от прошлого примера мало. Также идет создание потока, чтение файлов, выполнение функции из плагина и создание файлов в директории, что мы указали.
Вторая задача чуть интересней. Как я уже сказал, она смотрит за файлами и реагирует на их изменения. С помощью функции watch мы сначала указываем путь, который будет отслеживаться, а потом те задачи, что должны будут выполняться, в нашем случае это компиляция файла.

Давайте запустим сначала просто задачу gulp compile-sass. Как вы можете видеть у вас создалась директория dist/styles и внутри появился файл с расширением css и gulp прекратил свое выполнение.

Теперь если мы запустим задачу watch, то мы увидим, что gulp чего то ждет. Давайте поменяем что нибудь в файле main.scss. Во первых файл перекомпелировался, а во вторых у нас изменения отразились в файле.

Четвертый пример

Самый масштабный пример из всех, что я вам сегодня покажу. Мы с вами сейчас реализуем такие таски как: компиляция sass файлов, минификация изображений, конкатенация скриптов и

запуск вебсервера.

Давайте начнем с необходимой структуры файлов.

В папке src создадим еще папку images (пока оставим ее пустой), папку scripts. Внутрь папки scrips занусем два файла scripts1.js и scripts2.js. Со следующим содержанием соответсвенно

```
console.log('Script 1');
```

```
console.log('Script 2');
```

Файл index.html вынесем в корень проекта, то есть на тот уровень, где лежит gulpfile.js.

А также приведем его к следующей структуре:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="dist/styles/main.css">
  <title>Gulp Project</title>
</head>
<body>
  <h1>Hello, Gulp!</h1>
  <h1>Hello world!</h1>
  <p>Привет мир!!!!!</p>
  <img src="dist/images/images.jpg" alt="">
  <script src="dist/scripts/main.js"></script>
</body>
</html>
```

Далее доставим необходимые библиотеки

```
npm install gulp-imagemin@7.1.0
```

```
npm install gulp-concat
```

```
npm install gulp-webserver
```

Теперь напишем новый gulpfile.js

```
const gulp = require('gulp');
const sass = require('gulp-sass')(require('sass'));
const imagemin = require('gulp-imagemin');
const concat = require('gulp-concat');
const webserver = require('gulp-webserver');

gulp.task('compile-sass', function() {
  return gulp.src('src/styles/*.scss')
    .pipe(sass())
    .pipe(gulp.dest('dist/styles'));
});

gulp.task('minify-images', function() {
  return gulp.src('src/images/*')
    .pipe(imagemin())
    .pipe(gulp.dest('dist/images'));
});

gulp.task('concat-scripts', function() {
  return gulp.src('src/scripts/*.js')
    .pipe(concat('main.js'))
    .pipe(gulp.dest('dist/scripts'));
});

gulp.task('webserver', function() {
  return gulp.src('./')
    .pipe(webserver({
      livereload: true,
      open: true
    }));
});

gulp.task('watch', function() {
  gulp.watch('src/styles/*.scss', gulp.series('compile-sass'));
  gulp.watch('src/images/*', gulp.series('minify-images'));
  gulp.watch('src/scripts/*.js', gulp.series('concat-scripts'));
});

gulp.task('default', gulp.series('compile-sass', 'minify-images', 'concat-scripts', 'webserver',
```

Данный gulpfile позволяет нам и компилировать файлы и наблюдать за изменениями в режиме реального времени. Давайте пройдемся по всем таскам.

compile-sass - повторяет ранее написанную

minify-images - сжимает изображения и помещает их в папку

concat-scripts - собирает все скрипты и помещает их в один файл

webserver - вебсервер для просмотра изменений в коде в режиме реального времени

watch - наблюдение за задачами и их запуск в случае обнаружения каких то изменений

default - запуск задач из публичного пространства

Давайте поиграемся с запуском. Запустим сначала только вебсервер к примеру... (поиграли). Теперь запустим watch (веб сервера не будет), теперь запустим все сразу и поиграемся с файлами.

Вот такой какой то этот gulp теперь Дмитрий расскажет вам про grunt