gulpfile-ninecms Documentation

George Karakostas

Jun 18, 2019

Contents

Front-end package management.

CHAPTER 1

Features

The module uses gulp to pack front-end static files.

It offers a number of common tasks which can be included in a gulpfile and used in any project regardless of the tech stack.

It also offers a default `gulpfile.js` and related dependencies in the `package.json`.

1.1 Installation

1.1.1 Node.js

Node.js is required. Make sure it is installed.

Useful links:

- https://nodejs.org/
- https://github.com/creationix/nvm
- https://github.com/Wtower/express-experiment#install-node

1.1.2 Gulp

Gulp is obviously required. Install with:

```
$ npm install -g gulp-cli
```

1.1.3 Imagemagick (optional)

Imagemagick is optional for optimizing static images in the relevant task:

```
$ sudo apt-get install imagemagick
```

1.1.4 gulpfile-ninecms

Finally, to install the project itself. Change to your project folder and:

```
$ npm install -D gulpfile-ninecms
```

1.2 How to use

A sample gulpfile.js is included in the project.

You can create new gulpfile or adapt this sample to your needs:

```
$ cp node_modules/gulpfile-ninecms/gulpfile.js .
```

1.2.1 Path configuration

The first section of the provided gulpfile is an example of a path configuration object. To customize, edit the gulpfile and the appropriate paths.

There can be multiple path configurations for multiple builds. For instance, the sample gulpfile provides a default path config for the main site and an `admin` path config for the administration interface. These two have separate builds so that the main site is not getting bloated with scripts and styles that are only needed for another part of the site.

For details on tasks, see tasks. Specify the appropriate paths of the following:

- `assets`: any *Assets* files that need to be copied unchanged, such as fonts or images.
- `less`: any *LESS* files to be compiled.
- `sass`: any *SASS/SCSS* files to be compiled.
- `css`: any CSS files to be processed.
- `js`: the JS entry file for browserify to be processed.
- `js_watch`: any JS files to be watched.
- `mocha`: any JS files to be tested.
- `images`: any images to be processed.
- `build`: the output directory. This is used in most task methods.
- `fonts`: the fonts output directory.

For example:

```
var paths = {
  assets: [],
  ...
};
```

1.2.2 Requirements

Include in your gulpfile:

```
var taskMethods = require('gulpfile-ninecms');
```

This deals with gulp requirements regarding the tasks themselves. Also include any additional requirements such as `gulp` and `argv` that gulpfile itself may use.

Alternatively, you can copy/paste the task methods from index.js into the gulpfile.

1.2.3 Task wire-up

Multiple tasks may run many times, each for every path configuration set. Gulp does not provide a way to provide a path parameter to task (relevant issue). Therefore, we need to 'wire-up' the task methods we use with the appropriate path into gulp tasks:

```
var tasks = {
  clean: function () { return taskMethods.clean(paths); },
  ...
};

gulp.task('clean', tasks.clean);
```

1.2.4 Execute

To build and watch:

```
$ gulp
```

To build only once (no watch), run:

```
$ gulp build
```

To build for production:

```
$ gulp build --production
```

1.3 Clean

1.3.1 Syntax

```
clean(paths)
```

1.3.2 Description

Delete the output path `paths.build`.

1.3.3 Dependencies

Should runs before all *build* tasks.

1.3.4 Example

```
var paths = {
  build: 'build/'
};

var gulp = require('gulp');
var argv = require('yargs').argv;
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  clean: function () { return taskMethods.clean(paths); }
};

gulp.task('clean', tasks.clean);

// for production we require the clean method on every individual task
var build = argv._.length ? argv._[0] === 'build' : false;
var req = build ? ['clean'] : [];

// individual tasks
gulp.task('...', req, ...);
```

1.4 Assets

1.4.1 Syntax

```
assets(paths)
```

1.4.2 Description

Copy the specified files or directories from `paths.assets` to `paths.build`.

This is useful when the build path is the same path that offers static files as images, and you wish to clean it entirely with clean.

This is also especially useful to copy static paths that are referred to by css files such as bootstrap. Use the `*` gulp feature for paths accordingly (see examples).

1.4.3 Example

```
var paths = {
  assets: [
    'node_modules/bootstrap/dist/*fonts/*',
    'static/*images/*'
  ],
```

(continues on next page)

(continued from previous page)

```
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  assets: function () { return taskMethods.assets(paths); }
};

gulp.task('assets', [], tasks.assets);
```

1.5 LESS

1.5.1 Syntax

```
less(paths)
```

1.5.2 Description

Compile LESS files from `paths.less` to `paths.build`. Maintains the same file names, but replacing the extention `.less` to `.css`.

The compilation procedure for `--production`:

- Compiles LESS
- Auto-prefixes styles
- Minifies compiled styles

The compilation procedure for non `--production`:

- Compiles LESS
- Builds source maps

1.5.3 Dependencies

The *CSS concatenate* task should run after less has finished, if the output from less is included in the css task.

1.5.4 Example

```
var paths = {
  less: [
    'private/stylesheets/*.less'
  ],
  build: 'build/'
};

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

(continues on next page)

(continued from previous page)

```
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  less: function () { return taskMethods.less(paths); }
};

gulp.task('less', [], tasks.less);
```

1.6 SASS/SCSS

1.6.1 Syntax

```
sass(paths)
```

1.6.2 Description

Compile SASS and SCSS files from `paths.sass` to `paths.build`. Maintains the same file names, but replacing the extention `.s?ss` to `.css`.

The compilation procedure for `--production`:

- Compiles SASS
- Auto-prefixes styles
- Minifies compiled styles

The compilation procedure for non `--production`:

- Compiles SASS
- Maintains comments
- Builds source maps

1.6.3 Dependencies

The *CSS concatenate* task should run after sass has finished, if the output from sass is included in the css task.

1.6.4 Example

```
var paths = {
  sass: [
    'private/stylesheets/*.s?ss',
    'private/javascripts/my-angular-app/**/*.s?ss'
  ],
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');
```

(continues on next page)

(continued from previous page)

```
var tasks = {
  sass: function () { return taskMethods.sass(paths); }
};

gulp.task('sass', [], tasks.sass);
```

1.7 CSS concatenate

1.7.1 Syntax

```
css(paths)
```

1.7.2 Description

Concatenate CSS files from `paths.css` into `paths.build /css/style.min.css`.

The concatenation procedure for `--production`:

- Concatenate CSS
- Auto-prefixes styles
- Minifies compiled styles

The compilation procedure for non `--production`:

- Concatenate CSS
- Builds source maps

1.7.3 Dependencies

The task should run after *LESS*/*SASS*/*SCSS* has finished, if the output from them is included in the css task.

1.7.4 Example

```
var paths = {
  css: [
    'node_modules/animate.css/animate.css',
    'build/stylesheets/main.css' // this is the output from LESS/SASS
  ],
  ...
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  less: function () { return taskMethods.less(paths); }
```

(continues on next page)

(continued from previous page)

```
  sass: function () { return taskMethods.sass(paths); }
  css: function () { return taskMethods.css(paths); }
};

gulp.task('less', [], tasks.less);
gulp.task('sass', [], tasks.sass);
gulp.task('css', ['less', 'sass'], tasks.css);
```

1.8 Browserify

1.8.1 Syntax

```
browserify(paths)
```

1.8.2 Description

Browserifies the specified `paths.js` file and outputs to `paths.build`.

If `--production` is specified it also uglifies the output script.

Use a single js entry point with all requires.

If `gulp watch` then the task also watches changes in the bundled requires.

1.8.3 Quick reference

Quick reference to browserifying common front-end packages.

Alternatively use the *JS concatenate* task.

Notation used below:

- npm package: Install the npm package
- index.js: Require the package in the entry js using the provided code
- gulpfile path: Additional paths to include in gulpfile build

jQuery

- npm package: jquery
- index.js: `$ = jQuery = require('jquery');`
- Using browserify with jquery
- jQuery plugins (eg. scrollTo)

Bootstrap

- npm package: bootstrap
- index.js: `require('bootstrap');`
- gulpfile assets path: `'node_modules/bootstrap/dist/*fonts/*'`
- gulpfile css path: `'node_modules/bootstrap/dist/css/bootstrap*(|-theme).css'`
- html5shiv: either ignore or leave as is
- Browserify with Bootstrap

Angular

- npm package: angular
- index.js: `require('angular');`
- Angular with browserify
- Any angular plugins such as angular-sanitize: the same as Angular.

Lightbox2

- npm package: lightbox2
- index.js: `require('lightbox2');`
- gulpfile css path: `'node_modules/lightbox2/dist/css/lightbox.css'`

animate.css

- npm package: animate.css
- gulpfile css path: `'node_modules/animate.css/animate.css'`
- No js file

wow.js

- dependencies: animate.css
- npm package: wow.js
- *Browserify-shim*
  - `package.json`:

```
"browserify": {"transform": [ "browserify-shim" ]},
"browser": {
  "wow": "./node_modules/wow.js/dist/wow.js"
},
"browserify-shim": {
  "wow": {"exports": "WOW"}
}
```

  - `index.js`:

```
var WOW = require('wow');
new WOW().init();
```

Owl carousel

- npm: owl.carousel
- gulpfile css path: `'node_modules/owl.carousel/dist/assets/owl.carousel.css'`
- provide any additional owl plugins css rules
- *Browserify-shim*
  - `package.json`:

```
"browserify": {"transform": ["browserify-shim"]},
"browser": {
  "owl.carousel": "./node_modules/owl.carousel/dist/owl.carousel.js"
},
"browserify-shim": {
  "owl.carousel": "$"
}
```

  - `index.js`:

```
$.fn.owlCarousel = require('owl.carousel');
```

Font awesome

- npm: font-awesome
- gulpfile assets path: `'node_modules/font-awesome/*fonts/*'`
- gulpfile css path: `'node_modules/font-awesome/css/font-awesome.css'`
- No js file

Normalize.css

- npm: normalize.css
- gulpfile css path: `'node_modules/normalize.css/normalize.css'`
- No js file

Swiper

- npm: swiper
- js: `require('swiper');`
- gulpfile css path: `'node_modules/swiper/dist/css/swiper.css'`

Puse icons

- npm: puse-icons-feather
- gulpfile asset path: `'node_modules/puse-icons-feather/*fonts/*'`
- No js file

1.8.4 Browserify-shim

Packages that are not CommonJS compatible require a relative entry in `package.json`, as described above. The following entry is also required:

```
"browserify": {
  "transform": ["browserify-shim"]
},
```

The browserify-shim package is already included in gulpfile-ninecms dependencies.

1.8.5 Example

```
var paths = {
  js: 'private/javascripts/index.js',
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  browserify: function () { return taskMethods.browserify(paths); }
};

gulp.task('browserify', [], tasks.browserify);
```

1.9 JS concatenate

1.9.1 Syntax

```
concatJs(paths)
```

1.9.2 Description

Concatenate JS files from `paths.js_watch` into `paths.build /js/index.min.js`.

The concatenation procedure for `--production`:

- Concatenate JS
- Uglifies the output script

The concatenation procedure for non `--production`:

- Concatenate JS
- Builds source maps

1.9.3 Dependencies

The task should run after *Preload Angular HTML* has finished, if the output from it is included in the js task.

1.9.4 Example

```
var paths = {
  js_watch: [
    'node_modules/jquery/dist/jquery.js',
    'node_modules/bootstrap/dist/js/bootstrap.js',
    // angular
    'node_modules/angular/angular.js',
    'node_modules/angular-resource/angular-resource.js',
    'node_modules/angular-animate/angular-animate.js',
    'node_modules/angular-sanitize/angular-sanitize.js',
    // angular app
    'private/javascripts/my-angular-app/*.module.js',
    'private/javascripts/my-angular-app/*.animations.js',
    'private/javascripts/my-angular-app/core/*.module.js',
    'private/javascripts/my-angular-app/core/*.filter.js',
    'private/javascripts/my-angular-app/core/**/*.module.js',
    'private/javascripts/my-angular-app/core/**/*.service.js',
    'private/javascripts/my-angular-app/my-angular-app*/*.module.js',
    'private/javascripts/my-angular-app/my-angular-app*/*.component.js',
    // other
    'private/javascripts/jquery.main.js',
    'build/partials.js'
  ],
  partials: [
    'private/*javascripts/my-angular-app/**/*.html'
  ],
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  concatJs: function () { return taskMethods.concatJs(paths); },
  preloadNgHtml: function () { return taskMethods.preloadNgHtml(paths); }
};

gulp.task('preloadNgHtml', req, tasks.preloadNgHtml);
gulp.task('concatJs', req.concat(['preloadNgHtml']), tasks.concatJs);
```

1.10 Preload Angular HTML

1.10.1 Syntax

```
preloadNgHtml(paths)
```

1.10.2 Description

Preloads Angular HTML templates specified in `paths.partials` into a generated partial JS file `paths.build/partials.js`.

Procedure:

- Minifies HTML templates
- Uses the ng-html2js module to generate `partials.js`.

Please note that the task extracts the Angular module name from the template directory name. Eg. for a template file:

```
private/javascripts/my-admin-app/my-admin-module/my-admin-module.template.html
```

the module name will be parsed as `myAdminModule`.

1.10.3 Dependencies

Run the task *JS concatenate* after this task has finished.

1.10.4 Example

```
var paths = {
  js_watch: [
    ...
    'build/partials.js'
  ],
  partials: [
    'private/*javascripts/my-angular-app/**/*.html'
  ],
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  concatJs: function () { return taskMethods.concatJs(paths); },
  preloadNgHtml: function () { return taskMethods.preloadNgHtml(paths); },
};

gulp.task('preloadNgHtml', req, tasks.preloadNgHtml);
gulp.task('concatJs', req.concat(['preloadNgHtml']), tasks.concatJs);
```

1.11 Lint JS

1.11.1 Syntax

```
lintJs(paths)
```

1.11.2 Description

Analyze `paths.js_lint` and output report to stdout.

Uses the ESlint library:

> Code linting is a type of static analysis that is frequently used to find problematic patterns or code that doesn't adhere to certain style guidelines.

It uses the default rules except:

- control characters (eg `\n`) are *not* required for file appends:

  ```
  'no-control-regex': 'off'
  ```

- allow double quotes to avoid escaping single:

  ```
  'quotes': ['error', 'single', {avoidEscape: true}]
  ```

- relax curly braces:

  ```
  'curly': ['error', 'multi-line']
  ```

1.11.3 Dependencies

The task should run after *Preload Angular HTML* has finished, if the output from it is included in the js task.

1.11.4 Example

```
var paths = {
  js_lint: [
    'private/javascripts/*.js',
    '*.js'
  ],
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  lintJs: function () { return taskMethods.lintJs(paths); },
};

gulp.task('lintjs', tasks.lintjs);
```

1.12 Nsp security

1.12.1 Syntax

```
nsp()
```

1.12.2 Description

Run Node Security (nsp) to help identify known vulnerabilities by npm packages that are used in the project's `package.json`. Outputs to stdout.

1.12.3 Example

```
var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');
gulp.task('nsp', tasksMethods.nsp);
```

1.13 Mocha tests

1.13.1 Syntax

```
preTest(paths)
mocha(paths)
```

1.13.2 Description

Run mocha tests for the files specified in `paths.mocha`.

Optionally run `preTest` before `mocha` to add istanbul coverage to the files specified in `paths.js_cover`.

This is mostly useful for Node.js projects and can be otherwise ignored.

The task outputs the results on stdout and also creates a coverage directory with HTML report.

1.13.3 Dependencies

`preTest` should have finished before `mocha` is run.

1.13.4 Example

```
var paths = {
  js_cover: [
    'models/*.js',
    'routes/**/*.js',
    'app.js'
  ],
```

(continues on next page)

(continued from previous page)

```
  mocha: [
    'spec/*.js',
    'spec/api/contact.stub.js',
    'spec/api/dashboard.js'
  ],
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  preTest: function () { return taskMethods.preTest(paths); },
  mocha: function () { return taskMethods.mocha(paths); }
};

gulp.task('preTest', tasks.preTest);
gulp.task('mocha', tasks.mocha);
gulp.task('test', ['preTest'], tasks.mocha);
```

1.14 Karma tests

1.14.1 Syntax

```
karma()
```

1.14.2 Description

Run karma tests based on a `/karma.conf.js` file.

This is mostly useful for front-end JS such as Angular apps.

The output is configured in the above conf file.

1.14.3 Example

```
var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');
gulp.task('karma', tasksMethods.karma);
```

1.15 Google web fonts

1.15.1 Syntax

```
fonts(paths)
```

1.15.2 Description

Download Google fonts specified in `paths.fonts /fonts.list` to `paths.build /fonts` and generate a stylesheet for them.

Uses gulp-google-webfonts.

New in v0.4.0: Now the `fonts.list` location needs to be specified in `paths.fonts`.

1.15.3 Example

In `gulpile.js`:

```
var paths = {
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  fonts: function () { return taskMethods.fonts(paths); }
};

gulp.task('fonts', [], tasks.fonts);
```

In `index.html`:

```
<link rel="stylesheet" href="/build/fonts/fonts.css">
```

1.16 Base64 images

1.16.1 Syntax

```
base64(paths)
```

1.16.2 Description

Convert all files in `url()` in provided stylesheets from `paths.base64` into base64-encoded data URI strings. The output is concatenated into a `paths.build/css/base64.css` file.

Uses gulp-base64.

Please note that this is useful only for minor performance gains. Not useful for embedding images into emails as most clients do not accept data uri.

1.16.3 Example

```
var paths = {
  base64: ['build/css/built_from_sass.css'],
  build: 'build/'
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  base64: function () { return taskMethods.base64(paths); },
};

gulp.task('base64', req.concat(['sass']), tasks.base64);
```

1.17 Inline CSS in HTML

1.17.1 Syntax

```
inlineCss(paths)
```

1.17.2 Description

Convert all css styles of an HTML file specified in `paths.inlineCss.html` into inline styles. Outputs to `paths.inlineCss.build`. Useful for emails.

Uses gulp-inline-css.

1.17.3 Example

```
var paths = {
  inlineCss: {
    html: 'private/html/email_body.html',
    build: 'templates/envelope/'
  }
};

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  inlineCss: function () { return taskMethods.inlineCss(paths); }
};

gulp.task('inlineCss', req.concat(['sass']), tasks.inlineCss);
```

1.18 Image optimisation

1.18.1 Syntax

```
images(paths, images)
clean_image_opts(paths)
```

1.18.2 Description

Create optimized images as specified in `images` object into `paths.images`.

Use `clean_image_opts` to delete previously created image styles in `paths.images`. Be careful that paths specified are correct before running.

Uses gulp_image_resize to create the images into a specified sub-directory of the original path.

1.18.3 Example

```
var paths = {
  images: 'media/'
};
var images = [
  {
    build: '/thumbnail',
    src: ['media/*/image/*.+(jpg|png)'],
    width: 150
  }, {
    build: '/thumbnail_crop',
    src: ['media/*/image/*.+(jpg|png)'],
    width: 150,
    height: 150,
    crop: true
  }, {
    build: '/thumbnail_upscale',
    src: ['media/*/image/*.+(jpg|png)'],
    width: 150,
    height: 150,
    upscale: true
  }, {
    build: '/gallery_style',
    src: ['media/*/image/*.+(jpg|png)'],
    width: 400,
    height: 1000
  }, {
    build: '/blog_style',
    src: ['media/*/image/*.+(jpg|png)'],
    width: 350,
    height: 226,
    crop: true
  }, {
    build: '/large',
    src: ['media/*/image/*.+(jpg|png)'],
    width: 1280,
    height: 1280
```

(continues on next page)

(continued from previous page)

```
  }
];

var gulp = require('gulp');
var taskMethods = require('gulpfile-ninecms');

var tasks = {
  images: function () { return taskMethods.images(paths, images); },
};

gulp.task('images', [], tasks.images);
```

1.19 Other information

1.19.1 Background

Initially developed for sites built with NineCMS. As of v0.6.0 the previous (totally basic) package management was removed. Relevant issue: django-ninecms#56.

Several other possibilities have been evaluated and rejected:

- django-bower
- django-pipeline
- django-compressor
- django-webpacks
- django-require
- django-grunt

1.19.2 Useful links

Useful links on setting up a gulpfile

- http://stackoverflow.com/questions/26273358/gulp-minify-all-css-files-to-a-single-file
- http://autoprefixer.github.io/
- https://github.com/gulpjs/gulp/blob/master/docs/recipes/using-multiple-sources-in-one-task.md
- https://github.com/isaacs/node-glob#glob-primer
- http://stackoverflow.com/questions/25038014/how-do-i-copy-directories-recursively-with-gulp
- https://scotch.io/tutorials/automate-your-tasks-easily-with-gulp-js
- http://stackoverflow.com/questions/27399596/how-to-combine-postcss-autoprefixer-with-less
- http://stackoverflow.com/questions/25713118/how-to-perform-multiple-gulp-commands-in-one-task
- https://www.npmjs.com/package/gulp-image-resize
- http://stackoverflow.com/questions/30372637/getting-started-with-browserify-import-local-files

1.19.3 ENOSPC

In case that `gulp watch` fails with ENOSPC, increase the inotify watch limit:

```
$ echo fs.inotify.max_user_watches=16384 | sudo tee -a /etc/sysctl.conf && sudo␣
↪sysctl -p  # bash
> echo fs.inotify.max_user_watches=16384 | sudo tee -a /etc/sysctl.conf; and sudo␣
↪sysctl -p  # fish
```

1.19.4 IDEs

PyCharm has a gulp plugin.