Automating image work

Since we are unlikely to compress images and less likely to create the individual images necessary to do a good work with responsive images, we have to leverage technologies to do it for us.

I use Gulp to build sites so I've incorporated image processing into my development and production pipeline.

There are two tasks for working with images in Gulp. I will cover them separately and explain the decisions I've made about them.

Install and configure the plugins

All these plugins are installed with Node. Like we do in all Node projects we first initialize an empty `package.json`. The `--yes` flag will accept all default parameters.

```
npm init --yes
```

We then instalal the packages that we'll use in the tasks.

```
npm i -D gulp \
gulp-imagemin \
imagemin-mozjpeg \
imagemin-webp \
gulp-responsive
```

Once we have initialized our package.json and installed all the installed all the packages that we want to use, it's time to start work in the Gulp configuration

The rest of the code in this post lives in a `gulpfile.js` file.

At the very top we import Gulp and the plugins we want to use using the CommonJS `require` syntax.

```
// Imagemin and Plugins
const gulp = require('gulp');
const imagemin = require('gulp-imagemin');
const mozjpeg = require('imagemin-mozjpeg');
const webp = require('imagemin-webp');
const responsive = require('gulp-responsive');
```

Process Images

The first function is to compress images using Imagemin and the gulp-imagemin plugin.

This will handle SVG, GIF, PNG, JPG and WebP images.

```
unction processImages() {
  return gulp.src('src/images/originals/**')
  .pipe(imagemin([
    imagemin.gifsicle({interlaced: true}),
      imagemin.optipng({optimizationLevel: 5}),
      imagemin.svgo({
          plugins: [
              {removeViewBox: true},
              {cleanupIDs: false}
      ]}),
    use: [
      mozjpeg(),
      webp({quality: 85})],
  ]))
  .pipe(gulp.dest('src/images'))
}
```

Although not strictly necessary, I chose to be explicit as to what plugins I'm using and what parameters each of hese plugins uses.

Gifsicle, Optipng and SVGO are bundled with Imagemin so we can just use them as is. Mozjpeg and WebP are additional libraries that add functionality in addition to or instead of the default bundled libraries.

Generate Responsive Images

The next step is to generate a set of responsive images.

The idea is that for each JPG and PNG image we will generate a set of responsive images. I've reduced the code example below to a single size with 2 different resolutions.

Device Pixel versus Logical Pixel

Before we jump into the code we need to look at the difference between logical pixels and device pixels since this will affect the way we generate our responsive images and what types of images we want to use for our originals.

Let's assume that we have an iPhone X with a resolution of 2436 x 1125 device pixels and Device Pixel Ratio (DPR) of 3.

> The DPR is defined by the device manufacturer. Simply put, it refers to the number of physical pixels contained in one logical pixel. For example, a device with a DPR of 2 means that one logical pixel contains 4 (2 x 2) physical pixels. Similarly, a DPR of 3 implies that a single logical pixel is equivalent to 9 physical pixels. from A Guide to Responsive Images on the Web

If we divide the height and width by the DPR We get the logical size of the screen in this case: 812 x 375 and that's why the following CSS would work:

```
.example {
  width: 400px;
}
```

We're telling the CSS parser to use 400 logical pixels instead of 400 device pixels.

The code

The code follows standard Gulp practices.

We first select the `src` for the task. In this case we've chosen all the JPG and

PNG files under `src/images/hires`.

We pipe the input through `gulp-responsive` to generate multiple versions of each image.

I've chosen to demonstrate one image into DPR resolutions and three formats, JPEG, PNG, and WebP.

The @2x images are using double the number of pixels to accomodate higher DPR values. When/if we wanted to add a 3x image we can add it like we did the 2x image in the examples below.

```
function generateResponsive() {
return gulp.src([
    'src/images/hires/**/*.{jpg,png}',
  ])
  .pipe(gulp.responsive({
    '*': [{
    // image-small.jpg is 200 pixels wide
      width: 200,
      rename: {
   '*'// image-small.jpg is 200 pixels wide// image-small.jpg is 200 pixe
      },
    }, {
    // image-small@2x.jpg is 400 pixels wide       suffix: '-small@2x',
        extname: '.jpg',
      },
    }, {
    // image-small.png is 200 pixels wide
  '-small@2x'// image-small.png is 200 pixels wide'-small',
        extname: '.png',
      },
    }, {
    // image-small@2x.png is 400 pixels wide
      widt'-small'// image-small@2x.png is 400 pixels wideall@2x',
        extname: '.png',
      },
    }, {
    // image-small.webp is 200 pixels wide
```

```
      width: 200,
  '.png'// image-small.webp is 200 pixels wide    extname: '.webp',
      },
    }, {
    // image-small@2x.webp is 400 pixels wide
      width: 200 * 2,
 '.webp'// image-small@2x.webp is 400 pixels wide    extname: '.webp',
      },
    }, {
    // Global configuration for all images
    // The output quality for JPE'.webp'// Global configuration for all im
    // The output quality for JPEG, WebP
    // and TIFF output formats
      quality: 80,
      // Use progressive (interlace) scan
      // for JPEG and PNG output
      progressive: true,
      // Skip enalrgement warnings
      skipOnEnlargement: true,
      // Strip all metadata
      withMetadata: true,
    }],
  })
      .pipe(gulp.dest('dist/images')));
}
```

We might not need all the images but it's better to have them and no need them than need them and not have them.

Export the functions

The final step is to export the functions soo that Gulp can see them. It's no different thaan exporting functions in ES2015 modules.

```
exports.imagemin = processImages;
exports.responsive = generateResponsive;
```

This will enable us to run them as `gulp imagemin` and `gulp-responsive`.