# Compressing AVIF images with Gulp

Now that AVIF support is enabled by default in Chrome stable (85) and Firefox (behind a flag) we really should look at how to encode images to AVIF as part of a build workflow.

In my normal workflow, I use imagemin, gulp-imagemin, imagemin-mozjpeg, imagemin-webp, and, experimentally, imagemin-guetzli.

As of right now there is no Node package to compress AVIF images (either direct library manipulation or wrapper for the existing scripts) so we're left to explore other options.


Compress
Original image

Figure 1: Squosh compressing an image to AVIF

The first one is use a tool like Squoosh from the Google Chrome team. It is awesome to have the option to do it online but, once you go beyond a handfult of images, the manual process gets tedious and can hog your browser. As Jake Archibald points out in AVIF has landed:

> At an 'effort' of 2, it takes a good few seconds to encode. 'Effort' 3 is significantly better, but that can take a couple of minutes. 'Effort' 10 (which I used for images in this article) can take over 10 minutes to encode a single image.

So, what other options do we have?

If you're on a Macintosh machine, one option is to compile an AVIF encoder like `libheif` or `libavif` and the child_spawn module to create a task to run the encoder.

The first naive attempt looks something like this:

```
gulp.task('avif', function() {
    return require('child_process')'child_process'vifenc ./images/*.png')
})
```

The problem is that `avifenc` the encode tool for `libavif` expects all the images to be of the same dimensions as it will build an image sequence (animation) by default and that's not what I want. I want the tool to compress each image indepemently. It also expects an explicit output image name.

After much fidling and playing with options, this was the best I could come up with. It works, I tested it with a small number of JPG and PNG images and it produced AVIF images that opened in Chrome and Firefox, the only way we can actually test them.

The code follows these steps:

1. Select all the files under the image directory

2. For each file encode it using `avifenc`
3. Change the extension to `.avif` from whatever the original was
4. Put them in the destination directory, defined in the `destination` constant

```
const gulp = require('gulp');
const exec = require('gulp-exec');
const rename 'gulp-exec'gulp-rename');

gulp.task('avif', function() {
  const destination = './converted'
  return gulp.src('./images'avif'/*') // 1
    .pipe(exec((file) =>
        `avifenc ${file.path} ${destination}/${file.path}`
      )) // 2
    .pipe(rename({ extname: '.avif'})) // 3
    .pipe(gulp.dest(`${destination}`)) // 4
});
```

It is not perfect.

The task depends on an external library outside the Javascript/Node ecosystem. Depending on your OS and how comfortable you are with compiling and installing tools, it may be a breeze or not.

It relies on a certain level of 'magic' to work as intended. If you look at the `avifenc` declaration, it uses a hacky way to specify the destination of the compressed files.

I can't figure out if it's completely necessary but, since it does its job and the task works as intended, I won't worry too much about it but just document it here.

It relies on users doing the right thing. The AVIF compression library works with PNG and JPG images as input. Anytihng else will fail. I tried using a special glob in `gulp.src` to specify it should only take PNG and JPG images and ignore everything else but I couldn't. So, for now the task relies on you putting only JPG and PNG files in the source directory.

The task uses the default encoder (AOM 2.0) with all settings at their default values. Depending on how your version of `libavif` was compiled, you may have access to additional encoders that can produce better results.

The task doesn't address how to serve the images using `picture`-based responsive images. I described the process in A way to leverage new image formats.

The task will continue to evolve it may take time to figure out some of these issues. My hope is that it'll become unnecessary as people build AVIF support into existing tools.