Using Plugins

Gulp plugins are Node Transform Streams that encapsulate common behavior to transform files in a pipeline - often placed between src() and dest() using the .pipe() method. They can change the filename, metadata, or contents of every file that passes through the stream.

Plugins from npm - using the "gulpplugin" and "gulpfriendly" keywords - can be browsed and searched on the plugin search page.

Each plugin should only do a small amount of work, so you can connect them like building blocks. You may need to combine a bunch of them to get the desired result.

```
const { src, dest } = require('gulp');
const uglify = require('gulp-uglify');
const rename = require('gulp-rename');

exports.default = function() {
  return src('src/*.js')
    // The gulp-uglify plugin won't update the filename
    .pipe(uglify())
    // So use gulp-rename to change the extension
    .pipe(rename({ extname: '.min.js' }))
    .pipe(dest('output/'));
}
```

Do you need a plugin?

Not everything in gulp should use plugins. They are a quick way to get started, but many operations are improved by using a module or library instead.

```
const { rollup } = require('rollup');

// Rollup's promise API works great in an `async` task
exports.default = async function() {
  const bundle = await rollup({
    input: 'src/index.js'
  });
```

```
  return bundle.write({
    file: 'output/bundle.js',
    format: 'iife'
  });
}
```

Plugins should always transform files. Use a (non-plugin) Node module or library for any other operations.

```
const del = require('delete');

exports.default = function(cb) {
  // Use the `delete` module directly, instead of using gulp-rimraf
  del(['output/*.js'], cb);
}
```

Conditional plugins

Since plugin operations shouldn't be file-type-aware, you may need a plugin like gulp-if to transform subsets of files.

```
const { src, dest } = require('gulp');
const gulpif = require('gulp-if');
const uglify = require('gulp-uglify');

function isJavaScript(file) {
  // Check if file extension is '.js'
  return file.extname === '.js';
}

exports.default = function() {
  // Include JavaScript and CSS files in a single pipeline
  return src(['src/*.js', 'src/*.css'])
    // Only apply gulp-uglify plugin to JavaScript files
    .pipe(gulpif(isJavaScript, uglify()))
    .pipe(dest('output/'));
}
```

Inline plugins

Inline plugins are one-off Transform Streams you define inside your gulpfile by writing the desired behavior.

There are two situations where creating an inline plugin is helpful:

Instead of creating and maintaining your own plugin.

Instead of forking a plugin that exists to add a feature you want.

```
const { src, dest } = require('gulp');
const uglify = require('uglify-js');
const through2 = require('through2');

exports.default = function() {
  return src('src/*.js')
    // Instead of using gulp-uglify, you can create an inline plugin
    .pipe(through2.obj(function(file, _, cb) {
      if (file.isBuffer()) {
        const code = uglify.minify(file.contents.toString())
        file.contents = Buffer.from(code.code)
      }
      cb(null, file);
    }))
    .pipe(dest('output/'));
}
```