

# Automatización de tareas con



Álvaro Navarro  
26/02/2016

# Índice

1. Introducción
2. Gulp vs Grunt
3. Gulp API
4. Estructura de un gulpfile
5. Plugins
6. Otras herramientas
7. Referencias

# 1. Introducción

- Auge de nuevos frameworks javascript (**AngularJs**, **Backbone**, **Ember.js**, etc.)
  - Para el desarrollo de aplicaciones sobre **Node.js**
  - Se requieren herramientas para gestionar los proyectos
- Las arquitecturas basadas en estos frameworks necesitan un **build system**
- Un **build system** es una colección de tareas que automatizan un trabajo repetitivo
  - Análogo a Maven en Java
- Los componentes más comunes en un flujo típico de front-end son:
  - Gestores de paquetes
  - Preprocesadores (opcionalmente)
  - Herramientas para la construcción de tareas
- **Gulp** es una herramienta para la gestión y automatización de tareas
- El único requisito para utilizar **Gulp** es tener instalado **Node.js**



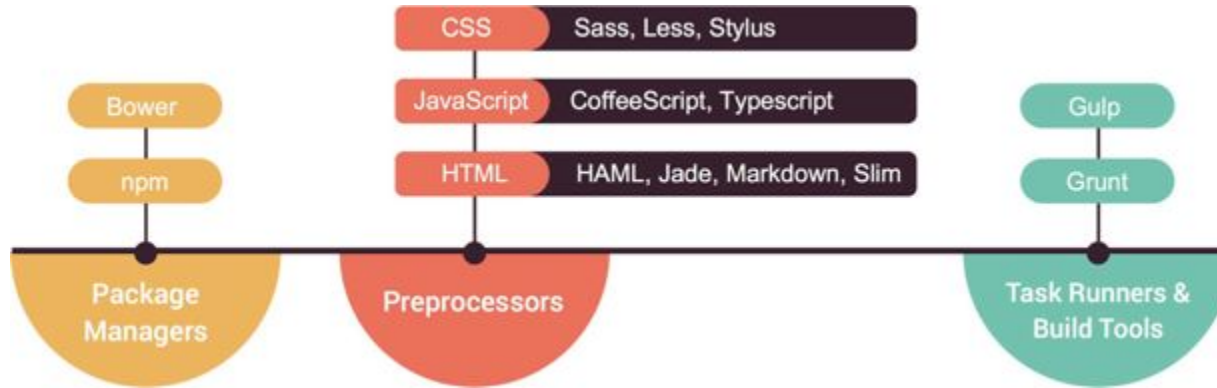
ANGULARJS



BACKBONE.JS



# 1. Introducción



- Estas herramientas standard construyen estructuras y tareas comunes para todos los desarrolladores
- El uso típico incluiría:
  - Compilación de CSS y JavaScript preprocesado
  - Concatenación
  - Minificación
  - Lanzar un servidor para la recarga automática en el browser
  - Creación de una **build** para despliegue

## 2. Grunt vs Gulp

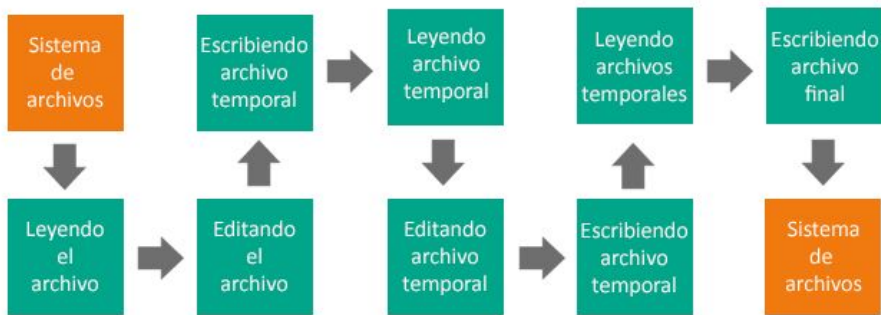
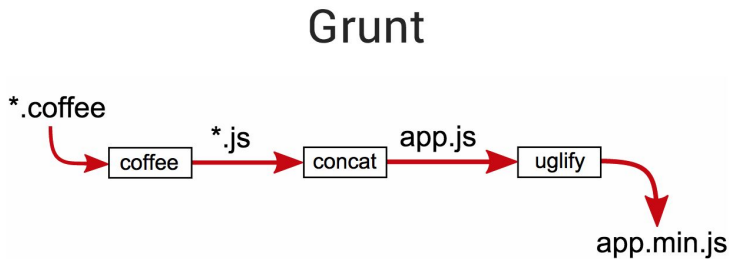
- Ambas son **task runners**: herramientas para definir y ejecutar tareas
- Estas tareas permiten automatizar la gestión del proyecto
- **Grunt** (2012) salió primero y **Gulp** (2014) después ofreciendo varias mejoras
- Tienen un enfoque distinto
  - Grunt se centra en la configuración
  - Gulp se centra en el código
- Internamente están implementados con conceptos distintos
  - Grunt almacena en ficheros temporales
  - Gulp utiliza streaming en memoria



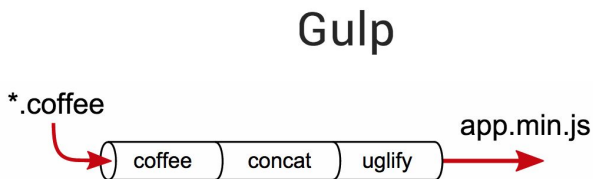


## 2. Grunt vs Gulp

- **Grunt** realiza lectura y escritura en ficheros durante el proceso



- **Gulp** usa streams, leyendo y escribiendo en memoria



## 2. Grunt vs Gulp

### Gulp Passes Files Through a Stream



- Un stream es un flujo de datos para el intercambio de información
- **Gulp** utiliza el módulo Stream de **Node.js**
  - Usa el paquete vinyl-fs para leer y escribir Streams
- Las tuberías (*pipes*) son un mecanismo para la comunicación y sincronización entre procesos
  - Basados en el patrón **productor/consumidor**
  - Están implementadas de forma muy eficiente en los sistemas operativos
  - Inician todos los procesos al mismo tiempo
  - Atienden automáticamente los requerimientos de lectura de datos para cada proceso cuando los datos son escritos por el proceso anterior



## 2. Grunt vs Gulp

Grunt - Configuración sobre código



```
module.exports = function(grunt, config) {
  'use strict';
  return {
    app: {
      src: [config.directories.app + '/index.html'],
      ignorePath: /\.\.\./bower_components\/
    },
    test: {
      devDependencies: true,
      src: 'test/karma.conf.js',
      ignorePath: /\.\.\./,
      fileTypes: {
        js: {
          block: /([\\s\\t]*)(\\\/\\s*bower:*(\\S*))?(\\n|\\r|.)*?(\\\/\\s*endbower)/gi,
          detect: {
            js: /\.*.js$/gi
          },
          replace: {
            js: '\\{filePath}\\',
          }
        }
      }
    }
  };
};
```

Gulp - Código sobre configuración





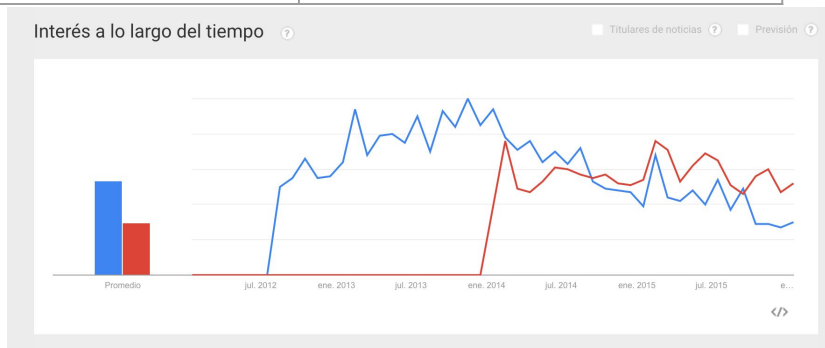
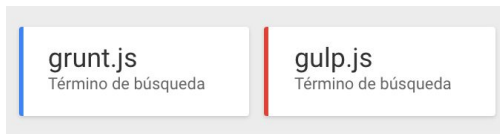
```
'use strict';

module.exports = function(gulp, config) {
  // Injecting the compiled files into the template
  gulp.task('wiredep', function() {
    gulp.src(config.karmaConf)
      .pipe(wiredep({
        devDependencies: true,
        dependencies: true
      }))
      .pipe(gulp.dest('./'));
  });
};
```

Ej: Tarea wiredep

### 3. Grunt vs Gulp

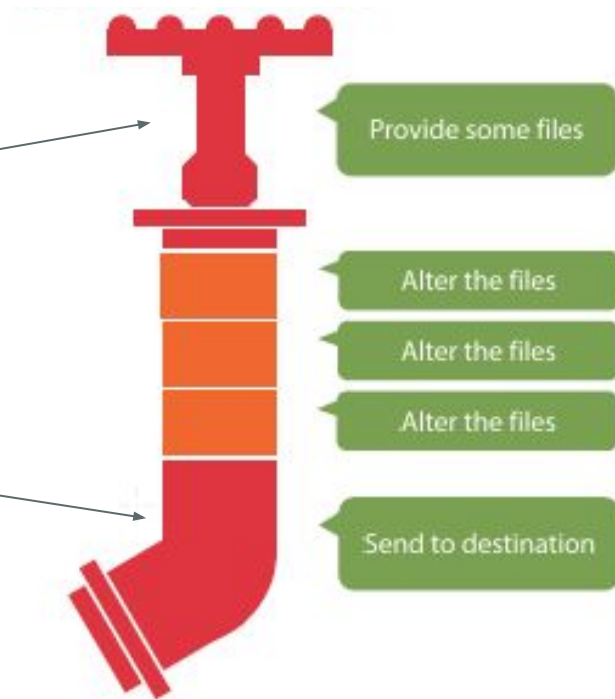
	 GRUNT	 GULP
Enfoque	Configuración sobre código	Código sobre configuración
Rendimiento	Basado en ficheros	Basado en streaming
Sintaxis	Más verboso	Más conciso y claro
Documentación	Gran comunidad de usuarios y documentación	Algo menor, aunque igualmente gran popularidad
Plugins	~5.525 plugins	~2166 plugins



# 3. Gulp API

1. `gulp.task`
  - Define una tarea
2. `gulp.src`
  - Lee ficheros
3. `gulp.dest`
  - Escribe los ficheros
4. `gulp.watch`
  - Observa si hay cambios en los ficheros

```
gulp.task('TaskName', [dependencyItem], function () {  
  return gulp  
    .src('./sourcePath')  
    .pipe(operation)  
    .pipe(gulp.dest('./destinationPath'));  
});
```



## 3. Gulp API

### 1. gulp.task()

- Define una tarea
- Tiene 3 argumentos:
  - el nombre de la tarea
  - dependencias de otras tareas
  - la función a ejecutar

```
gulp.task('TaskName', [dependencyItem], function () {  
    return gulp  
        .src('./sourcePath')  
        .pipe(operation)  
        .pipe(gulp.dest('./destinationPath'));  
});
```

### 2. gulp.src()

- Toma como parámetro una cadena que coincida con uno o más archivos
- Utiliza patrones que usa el intérprete de comandos de unix(shell)
- Retorna un stream que puede ser “pipeado” a un plugin adicional ó hacia el método gulp.dest()

### 3. gulp.dest()

- Canaliza y escribe archivos desde un Stream
  - Puede canalizar a varias carpetas
  - Creará las carpetas que no existan
  - Retornará el Stream, por si deseamos realizar alguna acción más
- Sirve para escribir los datos actuales de un Stream

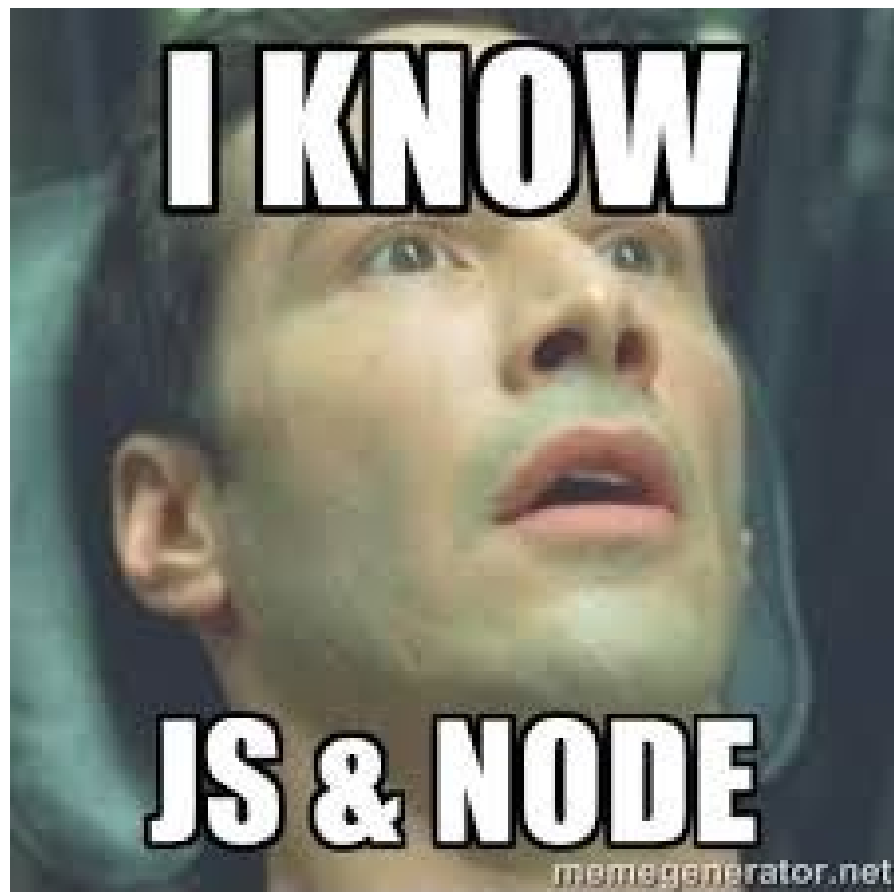
## 3. Gulp API

### 4. `gulp.watch()`

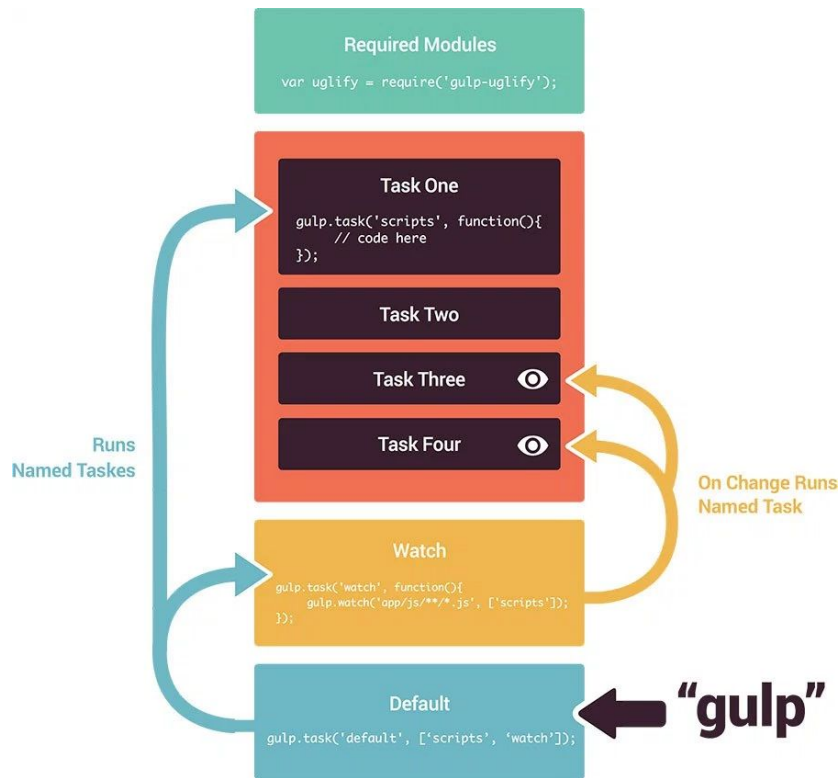
- Observa archivos y realiza una acción cuando se modifica un archivo
- Esto siempre devuelve un EventEmitter que emite los eventos de cambio
- Tiene 2 formas de uso:
  - **`gulp.watch(glob, [tasks])`**
  - **`gulp.watch(glob, callback)`**
- El primer parámetro es un glob con la ruta o un patrón con uno o varios ficheros
- El segundo puede ser una lista de tareas o una función a ejecutar

```
gulp.task('watch', function () {  
  gulp.watch('src/js/*.js', function(){  
    console.log('modifying js...');  
  });  
});
```

```
gulp.task('watch', function () {  
  gulp.watch('src/js/*.js', ['my-task']);  
});
```

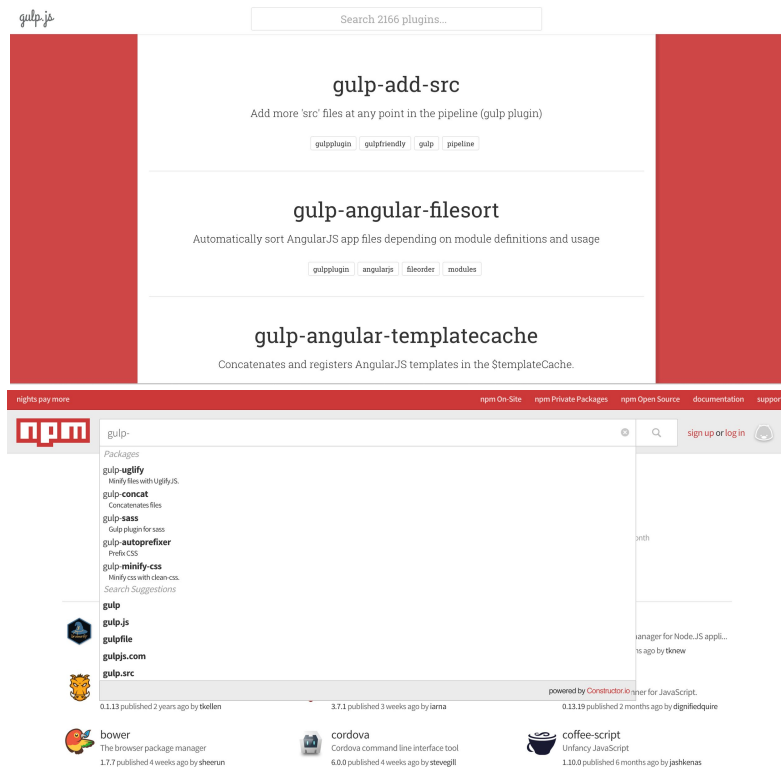


## 4. Estructura de un gulpfile



- El **gulpfile** debe de estar en la raíz del proyecto
- La estructura debe contener:
  - La importación de otros módulos
  - La importación de un fichero de configuración del proyecto (opcional)
  - La definición de las tareas
  - Observadores que se ejecutan en función de ciertos cambios (opcional)
  - Una tarea por defecto a ejecutar
- Si el **gulpfile** del proyecto conviene modularizarlo en ficheros con subtareas
- En una arquitectura conviene externalizar las tareas comunes en un módulo de **Node.js**

# 5. Plugins



- Gulp.js tiene alrededor de ~2166 plugins “oficiales” y otros no-oficiales
- Se pueden encontrar en
  - <http://gulpjs.com/plugins/>
  - <https://www.npmjs.com/>
- Los plugins disponibles permiten realizar tareas muy versátiles
  - Concatenar ficheros
  - Minimizar js
  - Minimizar imágenes
  - Inyectar dependencias dinámicamente
  - Ejecutar tests
  - Gestionar el código en un repositorio (svn, git)
  - Empaquetar un directorio
  - Ejecutar jshint, htmlhint
  - Generar css ejecutando sass, less, etc.
  - Mostrar ayuda



## 5. Plugins

- **gulp-concat**
  - Para juntar varios archivos en uno mismo.
  - Menos peticiones al servidor

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

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

- **gulp-uglify**
  - Sirve para minificar archivos js
  - Código se interpreta antes
  - Permite ejecutar una tarea de manera condicional

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

gulp.task('compress', function() {
  return gulp.src('lib/*.js')
    .pipe(uglify())
    .pipe(gulp.dest('dist'));
});
```

# 5. Plugins

- **gulp-zip**
  - Para comprimir directorios con ficheros
  - Permite generar una distribución front-end

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

gulp.task('default', () => {
  return gulp.src('src/*')
    .pipe(zip('archive.zip'))
    .pipe(gulp.dest('dist'));
});
```

- **gulp-if**
  - Permite ejecutar una tarea de manera condicional

```
var gulpif = require('gulp-if');
var uglify = require('gulp-uglify');

var condition = true; // TODO: add business logic

gulp.task('task', function() {
  gulp.src('./src/*.js')
    .pipe(gulpif(condition, uglify()))
    .pipe(gulp.dest('./dist/'));
});
```

## 5. Plugins

- **gulp-git**
  - Para ejecutar comandos sobre un repositorio de código GIT
  - Commit, add, push, clone, etc

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

// Run git commit without checking for a
// message using raw arguments
gulp.task('commit', function(){
  return gulp.src('./git-test/*')
    .pipe(git.commit(undefined, {
      args: '-m "initial commit"',
      disableMessageRequirement: true
    }));
});

// Run git push
gulp.task('push', function(){
  git.push('origin', 'master', function (err) {
    if (err) throw err;
  });
});
```

- **gulp con karma**
  - Podemos crear tareas utilizando librerías de nodejs que no son plugins de gulp

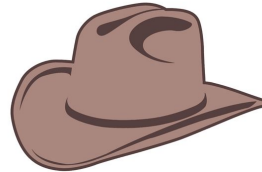
```
var gulp = require('gulp');
var Server = require('karma').Server;

/**
 * Run test once and exit
 */
gulp.task('test', function (done) {
  new Server({
    configFile: __dirname + '/karma.conf.js',
    singleRun: true
  }, done).start();
});
```

## 6. Otras herramientas

- Mimosa (2014)
- Broccoli (2014)
- Gruntjs (2012)
- Brunch (2011)
- Jake (2010)

Mimosa



# 7. Referencias

- <http://joellongie.com/gulp-build-system-fundamentals/>
- <https://medium.com/@preslavrachev/gulp-vs-grunt-why-one-why-the-other-f5d3b398edc4#.7zm1ho8ae>
- <https://habrahabr.ru/company/yandex/blog/239993/>
- <http://gruntjs.com/getting-started>
- <http://frontendlabs.io/1669--gulp-js-en-espanol-tutorial-basico-primeros-pasos-y-ejemplos>
- <http://www.codeproject.com/Articles/1064203/Introduction-Style-Guidelines-and-Automation-Featu>
- [https://es.wikipedia.org/wiki/Tuber%C3%ADa\\_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Tuber%C3%ADa_(inform%C3%A1tica))
- <https://www.npmjs.com>
- <http://nightdeveloper.net/plugins-utiles-gulp/>
- <https://osiux.ws/2014/12/utiles-plugins-para-gulp/>

