



M

E

A

N

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
germanux@gmail.com



Automatización de tareas con



GRUNT



The streaming build system

INDICE

- 1) ¿Cuándo automatizar?
- 2) Tareas repetitivas y automatizables
 - Comprobar la validez del código JS
 - Reducir el tamaño de ficheros
- 3) Automatización
- 4) Grunt: automatización por configuración
- 5) Gulp: automatización por programación

Tareas repetitivas y automatizables

Comprobar la validez del código JS

- JavaScript, al ser un lenguaje dinámico no tenemos una fase de compilación como tal que nos permita detectar errores
- Existen herramientas como **jshint** que al menos nos ayudan a detectar errores sintácticos y nos pueden avisar de potenciales problemas.
- JSLint es un analizador de código JavaScript creado por Douglas Crockford que nos permitirá mostrar puntos en lo que tu código no cumpla unas determinadas reglas establecidas de “código limpio”.

Tareas repetitivas y automatizables

Comprobar la validez del código JS

- Existen otras alternativas como Google Closure Linter o JavaScript Lint.
- Existe también alguna herramienta más como jQuery Lint que analiza tu código mientras se está ejecutando (modo Runtime) por lo que es necesario incluirlo como parte de tu código fuente en tu proyecto.

Tareas repetitivas y automatizables

Comprobar la validez del código JS

- JSLint no es una herramienta óptima ya que es bastante exhaustiva y da muchos falsos positivos.
- Además tiene muchos detractores que alegan que los criterios evaluados son bastante subjetivos según el punto de vista de su creador.
- Por todo ello, algunos desarrolladores crearon un fork llamado JSHint.

Tareas repetitivas y automatizables

Comprobar la validez del código JS

- El objetivo de JSHint es mejorar las mediciones que eran bastante arbitrarias en JSLint.
- Es frustrante ejecutar un código realizado por ti y ver como la herramienta menoscaba tu aplicación de una manera innecesaria.
- Hay que tener en cuenta que los desarrolladores utilizan diferentes estilos y convenciones y la herramienta debe adaptarse a ellos.
- El objetivo de JSHint es no imponer un convenio particular

Tareas repetitivas y automatizables

Comprobar la validez del código JS

- `npm install -g jshint`
- Para utilizar jshint desde node.js primero tenemos que instalarlo:
 - `npm install -g jshint`
- Y a continuación ejecutarlo sobre los ficheros que queramos:
 - `jshint file.js`

Tareas repetitivas y automatizables

Comprobar la validez del código JS

- Más información: <http://jshint.com/>
- Code Conventions for the JavaScript Programming Language :

<http://javascript.crockford.com/code.html>

Tareas repetitivas y automatizables

Reducir el tamaño de ficheros javascript con UglifyJS

- Una de las cosas más habituales antes de poner en producción una aplicación desarrollada con javascript, es unificar los ficheros javascript y reducir su tamaño para mejorar el tiempo de descarga (algunos dirán que también para ofuscarlos y que no sea fácil ver el código fuente).
- Hay varias herramientas para node.js que nos permiten hacer eso.

Tareas repetitivas y automatizables

Reducir el tamaño de ficheros javascript con UglifyJS

- Una de ellas es UglifyJS. Utilizarla es muy sencillo, la instalamos con:
 - `npm install -g uglify-js`
- Y podemos invocarla con:
 - `uglifyjs file1.js file2.js -o app.min.js`

Tareas repetitivas y automatizables

Reducir el tamaño de ficheros javascript con UglifyJS

Algunas opciones disponibles son:

- `--screw-ie8`
 - Utilice este indicador si no desea admitir Internet Explorer 6/7/8.
- `--support-ie8`
 - Utilice este indicador para admitir Internet Explorer 6/7/8.
- `-o, --output`
 - Archivo de salida (STDOUT predeterminado).
- `-e, --enclose`
 - Insertar todo en una función grande, con una lista configurable parámetro / argumento.
- `--comments`
 - Conservar los comentarios de copyright en la salida. De forma predeterminada, esto funciona como Google Closure, manteniendo los comentarios de estilo JSDoc que contienen "@license" o "@preserve".

Tareas repetitivas y automatizables

Reducir el tamaño de ficheros javascript con UglifyJS

Algunas opciones disponibles son:

- `--preamble`
 - Puede utilizarlo para insertar un comentario, por ejemplo, para obtener información sobre la licencia.
- `--stats`
 - Visualiza las operaciones en tiempo de ejecución en STDERR.
- `--wrap`
 - Insertar todo en una función grande, haciendo disponibles las variables "exportaciones" y "globales". Debe pasar un argumento a esta opción para especificar el nombre que su módulo tomará cuando esté incluido en, por ejemplo, un navegador.
- `--export-all`
 - Sólo se utiliza cuando `--wrap`, esto le dice a UglifyJS que agregue código para exportar automáticamente todos los globales.
- `--lint`
 - Mostrar algunas advertencias de alcance (ámbito o scope)
- `-v, --verbose`
 - Modo verboso

Más en
<https://github.com/mishoo/UglifyJS2>

Automatización

- Los automatizadores de tareas nos permiten juntar toda la ejecución de tareas relacionadas con nuestra aplicación, como:
 - Minificación
 - Compilación
 - Validación de sintaxis
 - Pruebas unitarias
 - Observar cambios de tus archivos
 - Concatenación de archivos
 - Copiado de archivos de una ruta a otra
 - Borrado de archivos
 - Generar documentación
 - Crear sprites, etc,

Automatización

- Una vez configurado, un corredor de tareas, este puede convertir todo ese trabajo o esfuerzo “manual” a prácticamente cero esfuerzo de tú parte.
- Esas tareas se harán automáticamente, en el orden que tú definas, bajo las instrucciones que tú le des.
- Con esto olvidate de hacer cosas “manuales”.

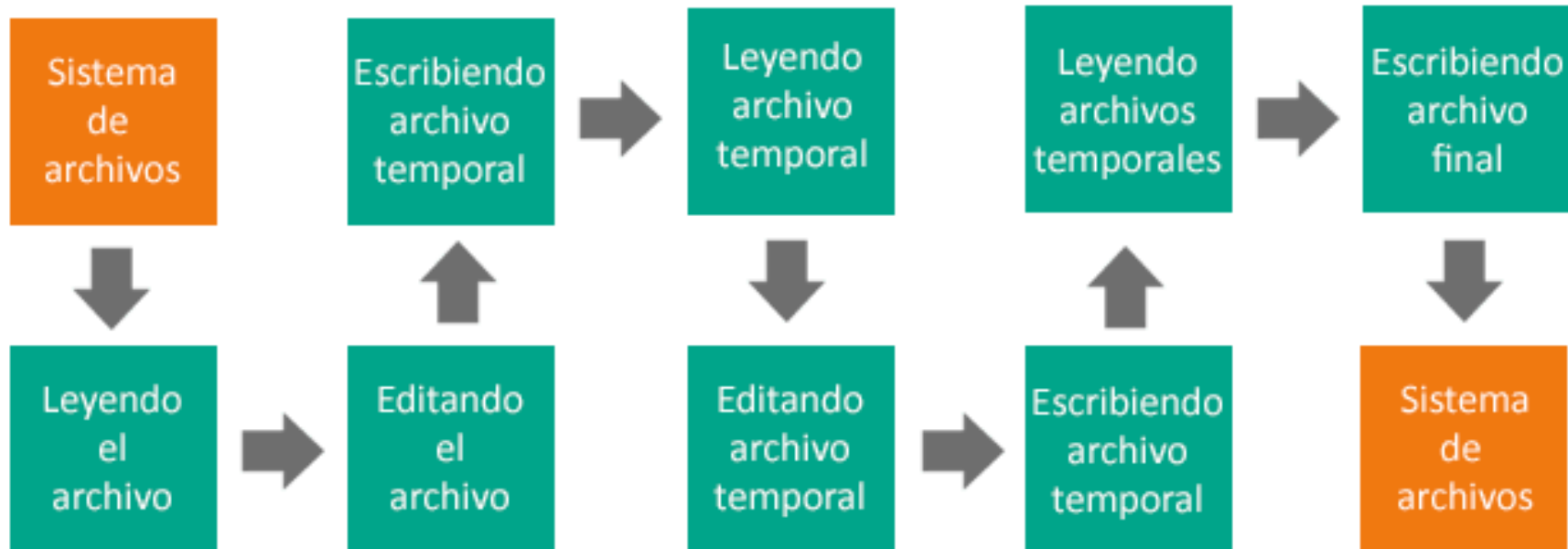
Grunt



- Grunt puede ejecutar casi cualquier tarea que puedas definir de manera programada, incluso gran parte de las tareas comunes conforman ahora la comunidad de plugins de grunt
- Entre ellos podrás encontrar minificadores y compresores de código, convertidores de un lenguaje a otro (como SASS a CSS), ejecución de pruebas, validadores, rutinas de conexión a servidores, instalación de dependencias especiales, y mucho más.

Grunt

- En la siguiente imagen veremos como Grunt.js manipula los archivos al realizar sus tareas:



Grunt

- Gruntfile
 - Este archivo escrito en Javascript (o CoffeeScript) le indica a Grunt qué tareas puede ejecutar sobre el proyecto y como ejecutarlas.
 - Inicialmente un Gruntfile tiene una estructura bastante sencilla

Grunt

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    propiedadExterna: ...,  
    tarea: {  
      options: {  
        // opciones especificas de tarea que sobrescribirán las por defecto.  
      },  
      destino: {  
        options: {  
          // opciones especificas de destino que sobrescribirán las de la tarea.  
        },  
        propiedad1: ...,  
        propiedad2: ...  
      }  
    }  
  });  
  
  grunt.loadNpmTasks('...');  
  grunt.registerTask('default', ['tarea']);  
}
```

Grunt

- Viéndolo a gran escala notaremos primero un envoltorio de la configuración lo cual permitirá ejecutar lo que esté dentro cuando ejecutemos el comando grunt.
- Luego adentrándonos en la configuración veremos atributos como propiedades externas que nos servirán para hacer referencia dentro de las tareas.
- Veremos la definición de una tarea sobre la cual se puede definir una serie de propiedades de opciones que puedan ser necesaria de especificar según sea el caso.

Grunt

- Cada tarea puede poseer uno o más destinos, los cuales se definirán como las acciones a tomar o ejecutarse para cada tarea.
- Es posible ejecutar destinos individuales de cada tarea o todos ellos secuencialmente según se necesite.
- Debido a que muchas de las tareas que ejecuta Grunt suelen involucrar manejo de archivos, este suele ser bastante flexible con la manera en que sus tareas son definidas.

Grunt

- En caso de que estemos utilizando un plugin debemos cargarlo con nuestra penúltima línea, este debe ser un módulo instalado por npm que contenga la tarea.
- Finalmente registramos la tarea para el comando por defecto de grunt.

Grunt

- En caso de que estemos utilizando un plugin debemos cargarlo con nuestra penúltima línea, este debe ser un módulo instalado por npm que contenga la tarea.
- Finalmente registramos la tarea para el comando por defecto de grunt.
- Veremos un ejemplo para entenderlo todo un poco mejor

Grunt

- Grunt.js y sus plugins se instalan y se gestionan a través de NPM, que es el gestor de paquetes de Node.js.
- Para empezar instalar la interfaz de línea de comandos de Grunt.js (CLI) a nivel global en tu equipo.
 - `npm install -g grunt-cli`
- Veremos Grunt en la práctica en las siguientes diapositivas

Grunt

- Primero crearemos un sencillo package.json con información como esta:

```
{  
  "name": "03-grunt-prueba",  
  "description": "Prueba de Grunt.js",  
  "version": "1.0.0",  
  "author": "Full Stack MEAN Developer Getafe",  
  "license": "GPL"  
}
```

Grunt

- Instalar también el módulo de Grunt y el plugin minificador de Javascript.
 - `npm install grunt --save-dev`
 - `npm install grunt-contrib-uglify --save-dev`
- La opción que hemos utilizado `--save-dev` indica que el módulo debe ser agregado como una dependencia de desarrollo al `package.json`:

Grunt

- package.json:

```
{
  "name": "03-grunt-prueba",
  "description": "Prueba de Grunt.js",
  "version": "1.0.0",
  "author": "Full Stack MEAN Developer Getafe",
  "license": "GPL",
  "devDependencies": {
    "grunt": "^1.0.1",
    "grunt-contrib-uglify": "^2.1.0"
  }
}
```

- Esto nos permitirá tan solo ejecutar **npm install** cuando queramos volver a instalar estas dependencias en otro ambiente.

Grunt

- Ahora procederemos a crear nuestro Gruntfile:

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    uglify: {  
      min: {  
        src: '<%= pkg.name %>.js',  
        dest: '<%= pkg.name %>.min.js'  
      }  
    }  
  });  
  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  
  grunt.registerTask('default', ['uglify']);  
};
```

Grunt

- Estructura básica:
 - El envoltorio o wrapping de la configuración.
 - Definición de la configuración de Grunt.
 - Una propiedad externa (pkg).
 - Una tarea (uglify).
 - Un destino dentro de la tarea (min).
 - Propiedades de ejecución de la acción.
 - Cargar un plugin que tenga el código de la tarea.
 - Registrar la tarea.

Grunt

- La propiedad externa `pkg` en este caso almacena el objeto compuesto por la configuración en el `package.json`, esto nos permite acceder a propiedades del mismo, como el autor, con tan solo llamarlo directamente con la notación de punto: `pkg.author`.
- En nuestro caso el valor de `pkg.name` se evaluaría a `prueba-grunt`.
- Las propiedades de ejecución del destino de minificación indican cual es la fuente y el destino del proceso.

Grunt

- Posteriormente, esos símbolos en las propiedades de ejecución (`<%= %>`), son parte del motor de plantillas de Grunt.
- Es decir, dentro de los símbolos podemos evaluar valores de variables tal como lo haríamos en otros motores de plantillas.
- Finalmente cargamos las tareas del plugin de minificación (uglify) y registramos nuestra tarea bajo la sección de tareas por defecto de Grunt.

Grunt

- Vigilar los cambios
- Si cada vez que cambiamos o modificamos algo de un js tenemos que volver a la consola para ejecutar la tarea, el proceso no es nada efectivo.
- Para evitar este paso, tenemos que hacerlo automático a través de otro plugin llamado **grunt-contrib-watch**.
- Este plugin se encargara de vigilar si estamos realizando cambios en algún archivo y en caso afirmativo ejecutará la tarea o tareas que le indiquemos.

Grunt

- La forma de instalarlo, sigue el mismo patrón que hemos visto antes, primero instalamos el plugin:
 - `npm install grunt-contrib-watch --save-dev`
- A continuación añadimos la tarea a Gruntfile.js:
 - `grunt.loadNpmTasks('grunt-contrib-watch');`

Grunt

- Y por último, incluimos la configuración de la tarea, previa consulta a la documentación del plugin.
- En nuestro caso queremos que cada vez que se modifique alguno de los archivos de la carpeta se vuelva a ejecutar la tarea uglify.

Grunt

- Para ello añadimos lo siguiente:

```
watch: {  
  scripts: {  
    files: ['./*.js'],  
    tasks: ['uglify'],  
    options: {  
      spawn: false,  
    }  
  } //scripts  
} //watch
```

- Cambiamos la tarea por defecto:

```
grunt.registerTask('default', ['watch']);
```

- Y volvemos a probar con **grunt**.

Gulp

- Gulp.js es un build system(sistema de construcción) open source que permite automatizar tareas comunes de desarrollo, tales como
 - la minificación de código JavaScript
 - recarga del navegador
 - compresión de imágenes
 - validación de sintaxis de código
 - y un sin fin de tareas más.
- Gulp.js prefiere el código sobre la configuración, esto no sólo lo hace muy fácil para escribir tareas, sino también lo hace más simple de leer y mantener.

Gulp

- Gulp.js tiene directrices estrictas para la creación de sus plugins, lo cual asegura que estos sean simples y que funcionen como se espera.
- Una de las directrices es la siguiente: “Un plugin sólo debe hacer una cosa y hacerla bien”.
- Esta directriz ha sido muy positiva para los usuarios de Gulp.js.

Gulp

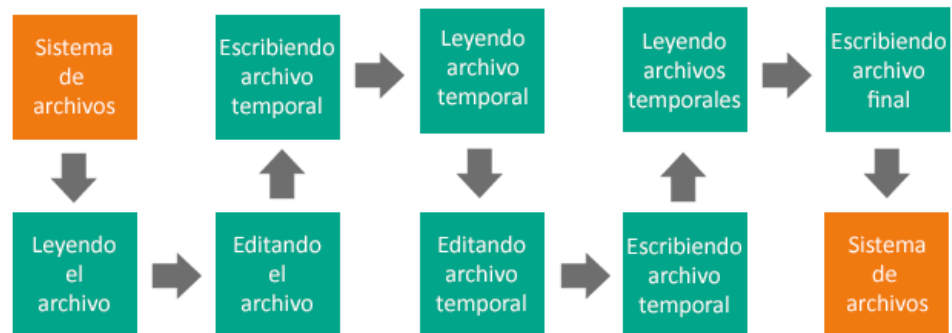
- Gulp.js utiliza el módulo Stream de Node.js, lo cual lo hace más rápido para construir, a diferencia de Grunt.js.
- Gulp.js no necesita escribir archivos y/o carpetas temporales en el disco duro, lo cual supone que realizará las mismas tareas que Grunt.js pero en menor tiempo.
- Gulp.js utiliza el método pipe(), este método obtiene todos los datos de un Stream legible(readable) y lo escribe en destino que le indiquemos ó de lo contrario lo entrega o sirve hacia otro pipe.

Gulp

- En la siguiente imagen veremos como Gulp.js manipula los archivos al realizar sus tareas:



Recordando Grunt:



Gulp

- Como podemos ver, aunque los 2 hicieron la misma tarea Gulp.js no escribió archivos temporales en el disco duro. Gulp.js realizó 4 operaciones y en cambio Grunt.js realizó 8 operaciones.

Gulp

- Instalación:
 - `npm install -g gulp`
- Verificamos que Gulp.js ha sido instalado correctamente.
 - `gulp -v`
- Si lo tenemos instalado correctamente, nos mostrará lo siguiente o algo parecido:

```
CLI version 3.8.6  
Local version undefined
```

Gulp

- Una vez instalado en nuestro sistema estamos listos para crear nuestro primer proyecto usando Gulp.js, nuestro pequeño proyecto concatenará dos archivos .js en uno solo y luego lo minificará.
- Así que configuraremos 2 tareas(concatenar y minificar), todo esto contenido en una tarea llamada “demo”.
- Creamos una carpeta llamada: 05_node_gulp,
- Ingresamos a esa carpeta mediante terminal.
- Dentro creamos nuestro archivo: gulpfile.js, que es el archivo que Gulp.js necesita para saber que tareas realizará.

Gulp

- Luego preparamos nuestro proyecto con
 - npm init
- Npm nos pedirá los datos de nuestro proyecto
- Con esto, Npm nos debe haber creado un archivo package.json:

```
{  
  "name": "05_node_gulp",  
  "version": "1.0.0",  
  "description": "Gulp: Primeros pasos",  
  "main": "gulpfile.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

Gulp

- Ahora agregaremos las dependencias de desarrollo a nuestro proyecto, la primera a instalar será: gulp
 - `npm install --save-dev gulp`
- Luego instalamos: gulp-concat
 - `npm install --save-dev gulp-concat`
- Y finalmente instalamos: gulp-uglify
 - `npm install --save-dev gulp-uglify`
- Recordad que sí no agregamos el parámetro: `--save-dev`, entonces Npm no agregará este paquete como una dependencia de desarrollo de nuestro proyecto y mucho menos lo agregará a nuestro archivo `package.json`.

Gulp

- Como podremos observar, nuestro archivo package.json a cambiado y debería contener algo parecido a lo siguiente:

```
{  
  "name": "05_node_gulp",  
  "version": "1.0.0",  
  "description": "Gulp: Primeros pasos",  
  "main": "gulpfile.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "gulp": "^3.9.1",  
    "gulp-concat": "^2.6.1",  
    "gulp-uglify": "^2.0.1"  
  }  
}
```

Gulp


- Creamos la carpeta **js** y dentro de esta carpeta crearemos la carpeta **source**.
- Dentro de la carpeta source crearemos el archivo **1.js** y le agregaremos el siguiente contenido:

```
// contenido del archivo 1.js
```

```
var sumar = function (a, b) {  
    return a + b;  
};
```

Gulp

- Dentro de la carpeta source crearemos el archivo 2.js y le agregaremos el siguiente contenido:



```
// contenido del archivo 2.js

var restar = function (a, b) {
    return a - b;
};
```

Gulp

- Primero para llevar a cabo las tareas que deseamos, requerimos los siguientes paquetes: gulp, gulp-concat y gulp-uglify, con require():
- Con el método gulp.task() definimos una tarea, este método toma 3 argumentos:
 - el nombre de la tarea,
 - la ó las tareas de las que depende esta tarea
 - y la función que ejecutará al llamar esta tarea.

Gulp

- Primero para llevar a cabo las tareas que deseamos, requerimos los siguientes paquetes: gulp, gulp-concat y gulp-uglify, con require():

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

Gulp

- Con el método `gulp.task()` definimos una tarea, este método toma 3 argumentos:
 - el nombre de la tarea,
 - la ó las tareas de las que depende esta tarea
 - y la función que ejecutará al llamar esta tarea.

```
gulp.task('demo', function () {  
    // Contenido de la tarea 'demo'  
});
```

Gulp

- El método `gulp.src()` toma como parámetro un valor glob es decir, una cadena que coincida con uno o más archivos usando los patrones que usa el intérprete de comandos de `unix(shell)` y retorna un stream que puede ser “pipeado” a un plugin adicional ó hacia el método `gulp.dest()`.
- Este parámetro puede ser una cadena o una colección(Array) de valores glob.

Gulp


- Ejemplos de globs:
 - `js/source/1.js` coincide exactamente con el archivo.
 - `js/source/*.js` coincide con los archivos que terminen en `.js` dentro de la carpeta `js/source`.
 - `js/**/*.js` coincide con los archivos que terminen en `.js` dentro de la carpeta `js` y dentro de todas sus sub-carpetas.
 - `!js/source/3.js` Excluye específicamente el archivo `3.js`.
 - `static/*.(js|css)` coincide con los archivos que terminen en `.js` ó `.css` dentro de la carpeta `static/`

Gulp

- Necesitamos encontrar todos los archivos que terminen en .js dentro de la carpeta js/source, así:
- `gulp.src('js/source/*.js')`
- Cada vez que Gulp.js encuentre un archivo que coincida con nuestro patrón, lo irá metiendo dentro de un Stream, que será como una colección de archivos.
- Siempre respetando las propiedades de cada archivo(ruta, etc).



Gulp

- Podemos decir que tendremos todos esos archivos con sus respectivas propiedades dentro de un Stream
 - Este Stream puede ser manipulado por Gulp.js.
- 

Gulp

- pipe()
 - El método pipe() puede leer, ayudar a transformar y grabar los datos de un Stream.
- gulp.dest()
 - Sirve para escribir los datos actuales de un Stream.
 - Canaliza y escribe archivos desde un Stream, por lo que puede canalizar a varias carpetas.
 - Creará las carpetas que no existan y retornará el Stream, por si deseamos realizar alguna acción más.

```
.pipe(gulp.dest('js/build/'))
```

Gulp

- El siguiente contenido de gulpfile.js queda:

```
/*
 * Dependencias
 */
var gulp = require('gulp'),
    concat = require('gulp-concat'),
    uglify = require('gulp-uglify');

/*
 * Configuración de la tarea 'demo'
 */
gulp.task('demo', function () {
    gulp.src('js/source/*.js')
        .pipe(concat('todo.js'))
        .pipe(uglify())
        .pipe(gulp.dest('js/build/'))
});
```


Gulp

- Con esto ya tenemos todo configurado
- Para ponerlo a prueba en nuestra terminal escribimos lo siguiente:
 - gulp demo
- Y si todo anda bien, nos dará el siguiente mensaje:

```
D:\PRONOIDE\CURSO_MEAN_STACK\repo_meanstack\07_node_express\05_node_gulp>gulp demo
[14:18:54] Using gulpfile D:\PRONOIDE\CURSO_MEAN_STACK\repo_meanstack\07_node_express\05_node_gulp\gulpfile.js
[14:18:54] Starting 'demo'...
[14:18:54] Finished 'demo' after 16 ms
```

Gulp

- Con esto ya tenemos todo configurado
- Para ponerlo a prueba en nuestra terminal escribimos lo siguiente:
 - gulp demo
- Y si todo anda bien, nos dará el siguiente mensaje:

```
D:\PRONOIDE\CURSO_MEAN_STACK\repo_meanstack\07_node_express\05_node_gulp>gulp demo
[14:18:54] Using gulpfile D:\PRONOIDE\CURSO_MEAN_STACK\repo_meanstack\07_node_express\05_node_gulp\gulpfile.js
[14:18:54] Starting 'demo'...
[14:18:54] Finished 'demo' after 16 ms
```

- El cual nos indica que la tarea demo se ejecutó con éxito en 16 milisegundos.

Gulp

- Para comprobar si se ejecutaron las 2 tareas requeridas, nos dirigimos a la carpeta
 - **js/build**
- y abrimos el archivo **todo.js** y nos debe mostrar el siguiente contenido:

```
var sumar=function(r,n){return r+n} ,  
restar=function(r,n){return r-n};
```

Gulp

- Una tarea también puede actuar como una lista de tareas
- Supongamos que queremos definir una tarea que corra otras 3 tareas por ejemplo: imágenes, css y js. Entonces escribiríamos lo siguiente:

```
gulp.task('estaticos', ['imagenes', 'css', 'js']);
```

- Lo que quiere decir que al ejecutar la tarea “estaticos” con el comando gulp estaticos se ejecutarán estas 3 tareas.
- Estas tareas correrán asíncronamente: correrán todas juntas al mismo tiempo sin ningún orden de ejecución.

Gulp

- Si deseamos que una tarea se ejecute sí y solo sí otra tarea haya terminado antes:

```
gulp.task('css', ['imagenes'], function () {  
  /*  
   * Aquí iría el contenido de mi tarea 'css'  
   * Que se ejecutará solo cuando la tarea  
   * 'imagenes' haya terminado.  
   */  
});
```

Gulp

- Si deseamos que una tarea se ejecute sí y solo sí otra tarea haya terminado antes:

```
gulp.task('css', ['imagenes'], function () {  
  /*  
   * Aquí iría el contenido de mi tarea 'css'  
   * Que se ejecutará solo cuando la tarea  
   * 'imagenes' haya terminado.  
   */  
});
```

- Entonces, cuando corramos la tarea “css”, Gulp.js ejecutará primero la tarea “imagenes”, esperará a que esta tarea termine y luego recién ejecutará la tarea “css”.

Gulp

- Gulp.js nos permite, al igual que Grunt, observar ficheros y cambios.
- `gulp.watch()`
 - Ver archivos y hacer algo cuando se modifique un archivo.
 - Esto siempre devuelve un EventEmitter que emite los eventos de cambio.
- Tiene 2 formas de usar:
 - `gulp.watch(glob, tareas)` ó `gulp.watch(glob, callback)`.

Gulp

- Ejercicio: Cambiar nuestra automatización usando `watch()` para que concatene y modifique automáticamente los ficheros .JS modificados
- Pistas:

```
gulp.watch('js/source/*.js', ['js']);
```

```
gulp.watch('js/source/*.js', function() {  
  /*  
   * Aquí iría el código de la acción que deseas realizar,  
   * Cuando hayan cambios en dichos archivos.  
   */  
});
```