eBook Gratuit

APPRENEZ gulp

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#gulp

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gulp

It is an unofficial and free gulp ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gulp.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec gulp

Remarques

Gulp est un système de compilation JavaScript, basé sur node.js comme Grunt. Il vous permet d'automatiser les tâches courantes pendant votre processus de développement. Gulp utilise les flux -concept et code-over-configuration pour un processus de construction plus simple et plus intuitif. Le concept de codage par configuration permet de créer des tâches plus lisibles et plus simples, tandis que les tâches fastidieuses sont très sur-configurées.

Versions

Version	Date de sortie
3.4	2014-01-17
3.7	2014-06-01
3.8	2014-06-10
3,9	2015-06-02
3.9.1	2016-02-09
4.0.0	2016-06-21

Examples

Installation ou configuration

1. Installez Node.js et NPM:

Gulp nécessite Node.js et NPM, le gestionnaire de paquets de Node. La plupart des installateurs incluent NPM avec Node.js. Reportez-vous à la documentation d'installation ou confirmez qu'elle est déjà installée en exécutant la commande suivante dans votre terminal,

```
npm -v
// will return NPM version or error saying command not found
```

2. Installez gulp globalement:

Si vous avez déjà installé globalement une version de gulp, lancez `npm rm --global gulp` pour vous assurer que votre ancienne version n'entre pas en conflit avec **gulp-cli** .

```
$ npm install --global gulp-cli
```

3. Initialisez votre répertoire de projet:

```
$ npm init
```

4. Installez gulp dans votre devdependencies de projet:

```
$ npm install --save-dev gulp
```

5. Créez un fichier gulpfile.js à la racine de votre projet:

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

gulp.task('default', function() {
  // place code for your default task here
});
```

6. Exécuter gulp:

```
$ gulp
```

La tâche par défaut s'exécutera et ne fera rien.

Pour exécuter des tâches individuelles, utilisez `gulp <task> <othertask>` .

Dépendance de la tâche

Vous pouvez exécuter des tâches en série en transmettant un second paramètre à `gulp.task()` .

Ce paramètre est un tableau de tâches à exécuter et à terminer avant l'exécution de votre tâche:

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

gulp.task('one', function() {
   // compile sass css
});

gulp.task('two', function() {
   // compile coffeescript
});

// task three will execute only after tasks one and two have been completed
// note that task one and two run in parallel and order is not guaranteed
gulp.task('three', ['one', 'two'], function() {
   // concat css and js
});
```

```
// task four will execute only after task three is completed
gulp.task('four', ['three'], function() {
   // save bundle to dist folder
});
```

Vous pouvez également omettre la fonction si vous souhaitez uniquement exécuter un ensemble de tâches de dépendance:

```
gulp.task('build', ['array', 'of', 'task', 'names']);
```

Note: Les tâches s'exécuteront en parallèle (toutes en même temps), donc ne supposez pas que les tâches vont commencer / finir dans l'ordre. À partir de gulp v4 , `gulp.series()` doit être utilisé si l'ordre d'exécution des tâches de dépendance est important.

Concat fichier js dans le sous-dossier en utilisant gulp

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

// include plug-ins
var uglify = require('gulp-uglify'),
    concat = require('gulp-concat');

// Minified file
gulp.task('packjsMin', function() {
   return gulp.src('node_modules/angular/*.js')
      .pipe(concat('script.js'))
      .pipe(uglify())
      .pipe(gulp.dest('Scripts'));
});

//Not minified file
gulp.task('packjs', function () {
    return gulp.src('node_modules/angular/*.js')
      .pipe(concat('allPackages.js'))
      .pipe(gulp.dest('Scripts'));
});
```

Documents CLI gulp

Les drapeaux

Gulp a très peu de drapeaux à connaître. Tous les autres indicateurs sont pour les tâches à utiliser si nécessaire.

- `-v` ou `--version` affichera les versions globales et locales de gulp
- `--require <module path>` nécessitera un module avant d'exécuter le fichier gulpfile. Ceci est utile pour les transpondeurs mais a également d'autres applications. Vous pouvez utiliser plusieurs drapeaux `--require`
- `--gulpfile <gulpfile path>` définira manuellement le chemin du fichier gulpfile. Utile si vous avez plusieurs fichiers gulpfiles. Cela définira également le CWD dans le répertoire gulpfile
- `--cwd <dir path>` définira manuellement le CWD. La recherche du fichier gulpfile, ainsi que la relativité de tous les besoins seront à partir d'ici

- `-T` ou `--tasks` affichera l'arborescence des dépendances de la tâche pour le fichier gulpfile chargé
- `--tasks-simple` affichera une liste de tâches en texte brut pour le fichier gulpfile chargé
- `--color` forcera les plugins gulp et gulp à afficher les couleurs même si aucun support de couleur n'est détecté
- `--no-color` forcera les plugins gulp et gulp à ne pas afficher les couleurs, même lorsque la prise en charge des couleurs est détectée
- `--silent` désactivera toute la journalisation gulp

La CLI ajoute process.env.INIT_CWD, cwd d'origine à partir duquel elle a été lancée.

Drapeaux spécifiques à la tâche

Reportez-vous à ce lien StackOverflow pour savoir comment ajouter des indicateurs spécifiques à une tâche.

les tâches

Les tâches peuvent être exécutées en exécutant `gulp <task> <othertask>` . Il suffit d'exécuter `gulp` pour exécuter la tâche que vous avez enregistrée, appelée `default` . S'il n'y a pas de tâche `default` , gulp sera une erreur.

Compilateurs

Vous pouvez trouver une liste des langues prises en charge à interpréter . Si vous souhaitez ajouter une prise en charge pour une nouvelle langue, envoyez des requêtes d'ouverture / des problèmes d'ouverture.

Lire Commencer avec gulp en ligne: https://riptutorial.com/fr/gulp/topic/1341/commencer-avec-gulp

Chapitre 2: Afficher les erreurs avec gulp-jslint

Examples

Installation et utilisation

Installation

```
$ npm install gulp-jslint --save-dev
```

Usage

Dans gulpfile.js ajoutez:

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

gulp.task('lint', function(){
    return gulp.src(['source.js'])
        .pipe(jslint({ /* this object represents the JSLint directives being passed down */
}))
        .pipe(jslint.reporter( 'my-reporter' ));
});
```

pour utiliser cette tâche:

```
$ ./node_modules/gulp/bin/gulp.js lint
```

Lire Afficher les erreurs avec gulp-jslint en ligne: https://riptutorial.com/fr/gulp/topic/7415/afficher-les-erreurs-avec-gulp-jslint

Chapitre 3: Chemin Gulp

Examples

Créer un chemin simple de Gulp vers l'application

Avant de `Node.JS` vous devez installer `Node.JS` et `npm`

La tête du Gulp.js est:

```
var gulp = require('gulp');
    clean = require('gulp-clean'); // A gulp plugin for removing files and folders.
    imagemin = require('gulp-imagemin'); // Minify PNG, JPEG, GIF and SVG images
```

Puis assigner le chemin des fichiers gulp

```
var bases = {
 app: 'app/',         // path to your app
 dist: 'dist/',       // path to the compiled application
};

var paths = {
 scripts: ['scripts/**/*.js', '!scripts/libs/**/*.js'],
 libs: ['scripts/libs/jquery/dist/jquery.js', 'scripts/libs/underscore/underscore.js',
'scripts/backbone/backbone.js'],
 styles: ['styles/**/*.css'],
 html: ['index.html', '404.html'],
 images: ['images/**/*.png'],
 extras: ['crossdomain.xml', 'humans.txt', 'manifest.appcache', 'robots.txt', 'favicon.ico'],
};
```

Exemple 1:

```
// Copy all other files to dist directly
    gulp.task('copy', ['clean'], function() {
 // Copy html
       gulp.src(index.html)
         .pipe(gulp.dest(bases.dist)); // same as .pipe(gulp.dest('dist'));
    });
```

Exemple 2:

```
// Imagemin images and ouput them in dist
gulp.task('imagemin', ['clean'], function() {
 gulp.src(paths.images, {cwd: bases.app})
 .pipe(imagemin())
 .pipe(gulp.dest(bases.dist + 'images/')); // same as .pipe(gulp.dest('dist/images/'));
});
```

Montre par défaut

```
// Define the default task as a sequence of the above tasks
gulp.task('default', ['clean', 'imagemin', 'copy']);
```

Lire Chemin Gulp en ligne: https://riptutorial.com/fr/gulp/topic/7759/chemin-gulp

Chapitre 4: Compression sans perte d'image (avec gulp-imagemin)

Syntaxe

1. imagemin ([plugins], {options})

Paramètres

Argument	La description
`sourcePath`	Répertoire source des images (par exemple: `/assets/images`)
`buildPath`	Chemin de destination (par exemple: `/static/dist/`)

Remarques

Le premier argument du constructeur `imagemin` est le tableau de plugins. Par défaut, les plugins suivants sont utilisés: `[imagemin.gifsicle(), imagemin.jpegtran(), imagemin.optipng(), imagemin.svgo()]`

Deuxième argument sont des options. Dans l'exemple ci-dessus, les options suivantes sont utilisées:

```
{
  progressive: true,
  interlaced: true,
  svgoPlugins: [{removeUnknownsAndDefaults: false}, {cleanupIDs: false}]
}
```

Ceux-ci sont complètement facultatifs.

`progressive` est utilisé par `imagemin-jpegtran` .

`interlaced` est utilisé par `imagemin-gifsicle` .

`removeUnknownsAndDefaults` et `cleanupIDs` sont utilisés par `imagemin-svgo` .

Examples

Installation et utilisation

Installation de dépendances (https://www.npmjs.com/package/gulp-imagemin)

```
$ npm install --save-dev gulp-imagemin
```

Usage

```
/*
 * Your other dependencies.
 */

var imagemin = require('gulp-imagemin');

/*
 * `gulp images` - Run lossless compression on all the images.
 */
gulp.task('images', function() {
  return gulp.src(sourcePath) // e.g. /assets/images
    .pipe(imagemin({
      progressive: true,
      interlaced: true,
      svgoPlugins: [{removeUnknownsAndDefaults: false}, {cleanupIDs: false}]
    }))
    .pipe(gulp.dest(buildPath + 'images')); // e.g. /static/dist/
});
```

Lire Compression sans perte d'image (avec gulp-imagemin) en ligne: https://riptutorial.com/fr/gulp/topic/6549/compression-sans-perte-d-image--avec-gulp-imagemin-

Chapitre 5: Créer un observateur

Examples

Tâche de surveillance

`config.paths.html` représente le chemin d'accès à votre fichier HTML.

```
gulp.task("watch", function() {
    gulp.watch(config.paths.html, ["html"]);
});
```

La tâche doit également être ajoutée à la valeur par défaut:

```
gulp.task("default", ["html", "watch"]);
```

Lire Créer un observateur en ligne: https://riptutorial.com/fr/gulp/topic/5026/creer-un-observateur

Chapitre 6: Créer une documentation avec gulp-jsdoc3

Examples

Installation

Tout d'abord, installez `gulp` et `gulp-jsdoc3` dans votre projet:

```
npm install gulp-jsdoc3 --save-dev
```

Puis ajoutez-le à votre gulpfile.js

```
var gulp = require('gulp');
var jsdoc = require('gulp-jsoc3');



gulp.task('doc', function (cb){
    gulp.src('src/*.js')
    .pipe(jsdoc(cb));
});
```

Pour documenter, par exemple, une fonction, vous devez ajouter un commentaire juste en haut de la fonction, comme ceci:

```
/**
 * @function example
 * @summary This is a short description of example
 * @author Whoever
 * @param {any} cb
 * @returns
 */
function example(cb){
    //Code
}
```

Si vous souhaitez en savoir plus sur les balises de bloc à utiliser, rendez- vous sur usejsdoc.org

Lire Créer une documentation avec gulp-jsdoc3 en ligne:
https://riptutorial.com/fr/gulp/topic/7365/creer-une-documentation-avec-gulp-jsdoc3

Chapitre 7: Fichiers concaténés

Examples

Concattez tous les fichiers CSS en un seul en utilisant gulp-concat

Tout d' abord, installer `gulp` et `gulp-concat` plug - in à votre projet localy

```
npm install --save-dev gulp gulp-concat
```

et ajouter la tâche gulp-concat à votre gulpfile.js

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

gulp.task('default', function() {
});

gulp.task('css', function() {
    return gulp.src('css/*.css')
        .pipe(concat('concat.css'))
        .pipe(gulp.dest('css/dist'));
});

gulp.task('default', ['css']);
```

Après le démarrage `gulp` commande, le plugin gulp-concat prendra tous les fichiers CSS situés dans le `css/` répertoire et les concaténer en un fichier `css/dist/concat.css`

Concat et Uglify JS et fichiers CSS

N'oubliez pas de npm pour installer tous les fichiers dans devDependencies en premier. Par exemple

```
npm install --save-dev gulp gulp-concat gulp-rename gulp-uglify gulp-uglifycss
```

Gulpfile.js

```
var gulp = require('gulp');
var gulp_concat = require('gulp-concat');
var gulp_rename = require('gulp-rename');
var gulp_uglify = require('gulp-uglify');
var uglifycss = require('gulp-uglifycss');

var destDir = './public/assets/dist/'; //or any folder inside your public asset folder
var tempDir = './public/assets/temp/'; //any place where you want to store the concatenated,
but unuglified/beautified files
//To concat and Uglify All JS files in a particular folder
gulp.task('js-uglify', function(){
    return gulp.src(['./public/js/**/*.js','./public/assets/js/*.js']) //Use wildcards to
```

```
select all files in a particular folder or be specific
        .pipe(gulp_concat('concat.js')) //this will concat all the files into concat.js
        .pipe(gulp.dest(tempDir)) //this will save concat.js in a temp directory defined above
        .pipe(gulp_rename('uglify.js')) //this will rename concat.js to uglify.js
        .pipe(gulp_uglify()) //this will uglify/minify uglify.js
        .pipe(gulp.dest(destDir)); //this will save uglify.js into destination Directory
defined above
});
//To Concat and Uglify all CSS files in a particular folder
gulp.task('css-uglify', function () {
  gulp.src('./public/assets/css/*.css') //Use wildcards to select all files in a particular
folder or be specific
  .pipe(gulp_concat('concat.css')) //this will concat all the source files into concat.css
        .pipe(gulp.dest(tempDir)) //this will save concat.css into a temp Directory
        .pipe(gulp_rename('uglify.css')) //this will rename concat.css into uglify.css, but
will not replace it yet.
    .pipe(uglifycss({
      "maxLineLen": 80,
      "uglyComments": true
    })) //uglify uglify.css file
    .pipe(gulp.dest(destDir)); //save uglify.css
});
```

Exécutez-les en suivant les commandes

```
gulp js-uglify
gulp css-uglify
```

Lire Fichiers concaténés en ligne: https://riptutorial.com/fr/gulp/topic/4398/fichiers-concatenes

Chapitre 8: Guide complet d'un flux de travail frontal avec Gulpjs 2 of 2

Remarques

Cool. Nous avons donc terminé notre automatisation du flux de travail.

Nous avons maintenant un fichier gulp, qui

- Responsabilise et réduit les images
- nettoie, autoprefixe, concatène et minimise Css
- Concatène et minimise JS
- Les montres pour les changements apportés aux actifs que ce soit HTML | CSS | JS et déclenche les tâches associées
- Crée un répertoire de génération et stocke tous les codes prêts à être utilisés dans le déploiement. Et tout cela, en arrière-plan, pendant que vous développez votre application.

Les références

Synchronisation du navigateur de séquence d' exécution

Examples

Configuration de la synchronisation du navigateur et configuration des observateurs pour le style et le script

REMARQUE

> **Cette page illustre l'utilisation de plugins gulp tels que browser-sync, gulp-watch et run-sequence, et continue à discuter de l' automatisation de workflows gulp depuis Gulpjs-workflow-automation-1 sur 2. Au cas où vous arriveriez ici, envisagez d'abord de** passer par ce poste.

- Tâche par défaut
- Tâche de surveillance - pour créer continuellement vos ressources prêtes à être déployées à la volée, quelle que soit l'image JS | css changements sur le cours du développement.

Commençons par la synchronisation par navigateur.

Tâche de chien de garde Gulp

Commençons par la tâche de surveillance.

Le but est de surveiller les changements que vous apportez en cours de développement. Toute modification devrait déclencher la tâche de gulp correspondante.

De plus, nous avons besoin d'une fonctionnalité qui synchronise vos modifications sur le navigateur.

Synchronisation du navigateur

Nous devons donc installer Browser Sync.

```
bash $ npm install browser-sync --save-dev
```

Avec cette prémisse, ouvrons notre gulpfile.js et ajoutons la fonctionnalité de surveillance. Exigeons la synchronisation du navigateur et définissons certaines variables pour utiliser ses fonctionnalités.

En haut du fichier gulp, ajoutez l'extrait ci-dessous. Placez-le juste en dessous des déclarations de compression d'image.

ainsi:

```
//Browser-sync

var sync = require('browser-sync').create();
var reload = sync.reload;
```

La synchronisation du navigateur avec votre navigateur sur le navigateur est une configuration simple. Créons une tâche appelée chien de garde.

ainsi:

```
$.task('watchdog', function() {


})
```

Maintenant, si nous parcourons les options de synchronisation du navigateur ici et que nous recherchons les paramètres du serveur, nous pouvons voir à quel point c'est facile.

Nous avons juste besoin de placer le ci-dessous à l'intérieur de notre tâche de surveillance

Snippet - 1 - À l'intérieur du chien de garde

```
/*
 Initiate Browser sync
 @documentation - https://www.browsersync.io/docs/options/
 */
 sync.init({
  server: {
```

```
    baseDir: "./"
},
port: 8000 //change it as required
});
```

Insérez le ci-dessus à l'intérieur de votre passe-partout chien de garde ci-dessus.

L'extrait suivant consiste à définir un observateur pour les styles, avec pour objectif de retraiter les fichiers CSS modifiés ou de nouveaux fichiers, et de déclencher automatiquement un rechargement du navigateur.

snippet - 2 - à l'intérieur du chien de garde

```
$.watch(['css/**/*', 'fonts/google/**/*.css'], reload).on('change', function(event) {
console.log(event.type + ':' + event.path)
if (event.type === 'deleted') {
uncache('styles', event.path);
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
sequence('optimizeStyles')
});
```

Insérez le ci-dessus à l'intérieur de votre passe-partout chien de garde ci-dessus.

Nous surveillons donc `" [fonts/google/**/*.css , /**/*.css ]"`

tous les fichiers css sous css / tous les fichiers css sous les polices / google / Lorsque quelque chose change ou qu'un nouveau fichier est ajouté, il déclenche la méthode de rechargement, qui est définie en haut de notre fichier gulpfile, dans la déclaration browsersync.

> Remarque: vous remarquerez peut-être qu'un gestionnaire d'événements **.on est** associé à l'observateur.

```
$.watch(['css/**/*', 'fonts/google/**/*.css'], reload).on('change', function(event)
```

Fondamentalement, tout élément CUD (Create | Update | Delete) déclenche la fonction de rechargement et transmet un objet Event comme paramètre à la fonction de rappel.

Le rappel est une fonction vitale, où nous pouvons réaliser des opérations telles que le non-cache sur la suppression des actifs.

- chemin
- type - Créer / Mettre à jour / Supprimer

Si un actif est supprimé, nous devons nous assurer que les caches que nous avons créés dans nos anciennes fonctions de minification, via gulp-caché et gulp-remember, nécessitent une mise à jour.

nous traitons cela dans l'extrait ci-dessous, qui se trouve dans le rappel sur le changement.

```
if (event.type === 'deleted') {
uncache('styles', event.path);
$$.remember.forget('auto-prefixed-stylesheets', event.path);
}
```

Remarque

$ -> alias pour gulp

$$ -> alias pour gulp-load-plugins

vous pourriez aussi remarquer que j'ai une `sequence('optimizeStyles');` après avoir écrit l'invocation non cachée

La méthode de séquence garantit que la méthode synchrone s'exécute par défaut dans un javascript asynchrone.

l'installer est simple

FAIRE

```
bash $ npm install run-sequence
```

ensuite, déclarez-le dans votre fichier gulp juste en dessous de la déclaration de synchronisation du navigateur.

```
var sequence = require('run-sequence');
```

Donc, avec cette compréhension, l'observateur des scripts est facile à écrire. juste des globes différents!

Donc, ajoutez cet extrait sous le style watcher dans le cadre de surveillance.

Snippet - 3 - intérieur de la tâche de surveillance

```
/*
on addition or change or deletion of a file in the watched directories
the change event is triggered. An event object with properties like
path,
event-type
is available for perusal passed to the callback

*/
$.watch('js/**/*', reload).on('change', function(event) {
console.log(event.type + ':' + event.path)
if (event.type === 'deleted') {
```

```
uncache('scripts', event.path);
$$.remember.forget('linted-scripts', event.path);
}
sequence('optimizeScripts');
});
```

Remarque

Nous avons utilisé deux fonctions dans nos extraits ci-dessus.

- non couché
- $$. Remember.forget Note:

$ -> Alias pour gulp

$$ -> Alias pour gulp-load-plugins

Définissons la fonction uncache quelque part dans notre fichier gulpfile.js, avant de l'appeler.

```
/*
Deletes a cache entry
*/
var uncache = function(cacheName, cacheKey) {
        var cache = $$.cached;
        if (cache.caches[cacheName] && cache.caches[cacheName][cacheKey])
            return delete cache.caches[cacheName][cacheKey];
        return false;
    }
    /*
    logs current cache created via gulp-cached
    */
var viewCache = function() {
    console.log($$.cached.caches)
}
```

Définition d'une tâche par défaut

Alors maintenant, laissez-nous terminer le code gulpfile, en définissant une tâche par défaut.

la tâche par défaut est celle qui s'exécute lorsque vous dites simplement

```
gulp
```

sur une invite de commande sous la racine de votre projet.

```
$.task('default', ['generateResponsiveImages'], function() {
 $.start('watchdog');
 console.log('Starting Incremental Build');
});
```

Lire Guide complet d'un flux de travail frontal avec Gulpjs 2 of 2 en ligne: https://riptutorial.com/fr/gulp/topic/6343/guide-complet-d-un-flux-de-travail-frontal-avec-gulpjs-2-of-2

Chapitre 9: Guide complet pour une automatisation du workflow frontal avec Gulpjs -1 of 2

Syntaxe

- npm install [nom-plugin] --save-dev
- npm install [nom du plugin] --save
- Fonction `<function-name>` (glob) {$ .src (glob) .pipe ([plug-in 1]). Pipe ([plug-in 2]) .... pipe ([plug-in]). Pipe ($ .dest (`<destination-name>`)}
- `$.task(<taskname> , [dependencies] , <body>);`

Remarques

Continue Reading - Création d'une tâche par défaut et configuration de la synchronisation du navigateur

Les références

- Gulp-Clean-Css
- Gulp-Uglify
- Gulp-Autoprefixer - pour les préfixes CSS qui maintiennent la compatibilité du navigateur
- Gulp-Cached
- Gulp-Remember
- Gulp-Jshint

Guide pas à pas sur l'automatisation des flux de travail Gulp pour les débutants absolus que j'ai documentés dans mon blog

Examples

Chargement de tous les plugins de Package.JSON

En supposant que vous compreniez comment installer Gulp, penchons-nous sur la nécessité d'exiger toutes les dépendances gulp de package.json dans le dossier racine de vos projets.

Si vous n'avez pas encore de fichier gulpfile, veuillez créer un fichier vide avec le nom

gulpfile.js

Tout d'abord, nous avons besoin de goûter. ainsi:

```
var $ = require('gulp');
```

Ensuite, nous allons charger tous nos plug-ins dans notre fichier gulpfich, par l'extrait ci-dessous

Remarque

S'il vous plaît lire les commentaires dans tous les extraits que nous allons inclure dans cette lecture, ils fournissent plus d'informations sur chaque fonctionnalité

```
/*
require gulp-load-plugins, and call it
gulp-load-plugins will require individually,
all the plugins installed in package.json at the root directory
these required plugins, are lazy loaded
that is, gulp-load-plugins will only require them when they are referenced in this file
plugins are available for perusal, via camelcased names, minus gulp
eg: gulp-clean-css is accessed by $$.cleanCss
*/
var $$ = require('gulp-load-plugins')();
```

REMARQUE

Dans tous les nombreux exemples que nous aurons dans l'article, nous alias

1. gulp comme $ et
2. gulp-load-plugins en tant que $$

uniquement pour faciliter la saisie.

Installation de plugins pour des images réactives | Css Minification | Js minification

Notre but est de

1. Rendez vos images conformes aux largeurs et aux échelles appropriées en générant une batterie d'images de largeurs variées et en les réduisant
2. Lint votre Javascript
3. Minimisez vos ressources - JS / CSS / HTML, vous permettant ainsi d'héberger un code plus léger que le code le plus léger
4. Regarder les fichiers css / js / image pour le changement et reconstruire les optimisations
5. Synchroniser vos modifications en cours de développement sur un navigateur servant votre site.

Nous avons besoin d'un certain nombre de plugins, alors installons-les tous. Veuillez exécuter tous ces éléments dans le dossier racine du projet.

Plugins de traitement d'images

```
bash $ npm install --save-dev gulp-responsive
bash $ npm install --save-dev gulp-imagemin
bash $ npm install --save-dev imagemin
bash $ npm install --save-dev imagemin-jpeg-recompress
bash $ npm install --save-dev imagemin-pngquant
```

Plugins d'optimisation d'actifs

```
bash $ npm install --save-dev gulp-clean-css
bash $ npm install --save-dev gulp-uglify
bash $ npm install --save-dev gulp-minify-html
bash $ npm install --save-dev gulp-jshint
bash $ npm install --save-dev gulp-concat
bash $ npm install --save-dev gulp-autoprefixer
```

Anatomie d'une fonction gulp

```
[Function <name>] (glob) {

$.src(glob)

.pipe([plugin 1])
.pipe([plugin 2])
.
.
.
.pipe([plugin n])
.pipe( $.dest(<destination-name>)

}
```

Remarque

pipe est une méthode qui diffuse tous les fichiers correspondant à l'entrée glob, à nos plugins (minifyhtml dans ce cas).

Il est simple de l'imaginer ainsi:

$ .src est ce qui construit le flux et les tubes de canalisation pour chaque fichier correspondant au glob vers le bas. Chaque plugin dans lequel le fichier est passé modifie son contenu en mémoire jusqu'à ce que $ .dest soit atteint / crée des fichiers diffusés par $ .src

Où ,

$ -> gulp

$$ -> gulp-load-plugins

Donc, avant d'écrire les fonctions d'optimiseur, nous devons installer deux plugins de mise en cache.

```
bash $ npm install --save-dev gulp-cached
bash $ npm install --save-dev gulp-remember
```

Vous pourriez vous demander pourquoi deux caches hein! **gulp-caché** , ne transmet que du contenu modifié ou nouveau dans le pipeline aux autres plug-ins. Donc, comme nous voulons que les fichiers sans changement soient utilisés pour concaténer dans un seul fichier par élément (css | js) également, nous avons besoin de **gulp-remember** en plus de gulp-caché

Nous utilisons d'abord **gulp-cached** pour construire une liste de fichiers qui ont changé

Deuxièmement, **n'oubliez** pas de garder en mémoire tous les fichiers transmis par cette liste.

Première exécution: aucun fichier n'est mis en cache, donc gulp-caché les transmettra tous à gulp-remember

Exécutions suivantes: seuls les fichiers modifiés ou nouveaux sont redirigés par gulp-caché. Comme le contenu du fichier actuel a changé, gulp-remember met à jour son cache.

Cool, laissez-nous écrire notre premier optimiseur

Optimiseur de style

```
// Goal

/*

1. cache existing files on the first run
2. each file ,
       a. Is autoprefixed with cross browser compatible prefixes for any css property (
justify-content for e.g)
       b. Is concatenated into app.css
3. app.css is minified
4. On subsequent runs , the above process is implemented on file modifications/additions only

*/

/*
*@src - input a glob pattern - a string eg 'images/**/*' or '**/*.css' or, an array eg
['glob1','glob2']
*/
var optimizeStyles = function(src) {

return $.src(src).
pipe($$.cached('styles')).
pipe($$.autoprefixer({
browsers: ['last 2 versions']
```

```
})).
pipe($$.remember('auto-prefixed-stylesheets')).
pipe($$.concat('app.css')).
pipe($.dest('build/css')).
pipe($$.cleanCss()).
pipe($$.rename({
suffix: '.min'
})).
pipe($.dest('build/css'))
}
```

Remarque

$ -> gulp

$$ -> gulp-load-plugins

$ .src -> construit les flux de fichiers correspondant au glob passé en src

$ .dest -> enregistre le fichier manipulé dans le chemin spécifié

Optimiseur de script

```
// Goal

/*

1. cache existing files on the first run
2. each file ,
       a. Is linted with jshint
       b. Is concatenated into app.js
3. app.js is minified
4. On subsequent runs , the above process is implemented on file modifications/additions only

*/

/*
*@src - input a glob pattern - a string eg 'js/**/*' or '**/*.js' or, an array eg
['glob1','glob2']
*/

var optimizeScripts = function(src) {

    return $.src(src).
    pipe($$.cached('scripts')).
    pipe($$.jshint()).
    pipe($$.jshint.reporter('default')).
    pipe($$.jshint.reporter('fail')).
    pipe($$.remember('linted-scripts')).
    pipe($$.concat('app.js')).
    pipe($.dest('build/js')).
    pipe($$.uglify()).
    pipe($$.rename({
        suffix: '.min'
    })).
```

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

}
```

Remarque

$ -> gulp

$$ -> gulp-load-plugins

$ .src -> construit les flux de fichiers correspondant au glob passé en src

$ .dest -> enregistre le fichier manipulé dans le chemin spécifié

Générer des images réactives

Passons maintenant au traitement de l'image. Donc, le but est d'avoir un tableau de tailles pour chaque image que vous allez servir.

Pourquoi?

> Eh bien, pour comprendre pourquoi nous avons besoin d'une batterie d'images avec une gamme de largeurs, nous devons réfléchir au fait qu'il y a probablement des milliards d'appareils avec des résolutions variées. Nous avons besoin d'une image à l'échelle sans trop de pixellisation. En même temps, nous devons améliorer les temps de chargement des pages en téléchargeant uniquement l'image, qui correspond à la largeur dans laquelle elle se trouve, et dont la dimension est également la plus petite possible. Il y a des blogs savants comme celui que Eric Portis a écrit, qui souligne l'inefficacité des requêtes de médias et sert de guide complet pour comprendre des concepts tels que srcsets et tailles.

Vous pouvez vous référer à l'épopée d'Eric Portis ici

Maintenant, notre fonction, doit prendre un glob, et une largeur comme entrées, et faire sa magie et pousser le fichier que chaque exécution génère, à une destination et réduire l'image réactivée.

Nous avons installé deux plugins de compression d'image dans notre premier exemple

Comme ces plugins **NE** commencent **PAS** par un préfixe "gulp-", nous devons les charger manuellement sur notre fichier gulpfile.

SO Exigeons-les manuellement, après la déclaration gulp-load-plugins en haut de notre fichier gulpfile.

ainsi:

```
var compressJpg = require('imagemin-jpeg-recompress');
```

```
var pngquant = require('imagemin-pngquant');
```

Il est intéressant de noter que gulp-responsive est livré avec un processeur d'image pointu, ce qui est mieux que imagemagick BY FAAAAR !. Sharp est ce que gulp-responsive utilise pour rogner vos images à la largeur désirée.

Vous pouvez regarder gulp-responsive-configuration-options pour une liste complète des paramètres de configuration. Je n'ai utilisé que

- width - pour rogner nos images à une largeur w, passé en paramètre
- rename - pour ajouter un suffixe au nom de l'image, de sorte qu'il reste unique

dans ma fonction de configuration ci-dessous. Ainsi, notre fonction va recadrer l'image à la largeur transmise en entrée, pour toutes les images correspondantes déchiffrées via l'entrée glob. Ensuite, chaque image est compressée en utilisant jpeg-recompress ou pngquant et enregistrée dans build / images.

Dans cet esprit, notre fonction serait comme suit:

```
/*
@generateResponsiveImages
*@Description:takes in a src of globs, to stream matching image files , a width,
*to resize the matching image to, and a dest to write the resized and minified files to
*@src - input a glob pattern - a string eg 'images/** /*' or 'images/*' or, an array
eg ['glob1','glob2']
*@return returns a stream
*/
var generateResponsiveImages = function(src, width, dest) {

    //declare a default destination
    if (!dest)
        dest = 'build/images';
    //case 1: src glob -  images/**/*
    // the base is the directory immediately preceeding the glob - images/ in this case
    //case 2: images/fixed/flourish.png : the base here is images/fixed/ - we are overriding
    // that by setting base to images.This is done so that, the path beginning after images/
    // - is the path under our destination - without us setting the base, dest would be,
    //build/images/flourish.png.But with us setting the base, the destination would be
    // build/images/fixed/flourish.png
    return $.src(src, {
        base: 'images'
    })

    //generate resized images according to the width passed
    .pipe($$.responsive({
            //match all pngs within the src stream
            '**/*.png': [{
                width: width,
                rename: {
                    suffix: '-' + width
                },
                withoutEnlargement: false,
            }],
            //match all jpgs within the src stream
            '**/*.jpg': [{
                width: width,
```

```
                rename: {
                    suffix: '-' + width
                },
                progressive: true,
                withoutEnlargement: false,
            }]
        }, {

            errorOnEnlargement: false,
            errorOnUnusedConfig: false,
            errorOnUnusedImage: false

        }))
        //once the file is resized to width, minify it using the plugins available per format
        .pipe($$.if('*.jpg', compressJpg({
            min: 30,
            max: 90,
            target: 0.5
        })()))
        //use file based cache gulp-cache and it will minify only changed or new files
        //if it is not a new file and if the contents havent changed, the file is served from
cache
        .pipe($$.cache($$.imagemin({
            verbose: true
        })))


    //write to destination - dest + path from base
    .pipe($.dest(dest));
}
```

Remarque

$ -> gulp

$$ -> gulp-load-plugins

$ .src -> construit les flux de fichiers correspondant au glob passé en src

$ .dest -> enregistre le fichier manipulé dans le chemin spécifié

Autres références

- Gulp-Responsive
- Gulp-Imagemin

Minificateur HTML

```
*
 *@minifyHtml
 *Description:takes in a glob src, and minifies all '.html' files matched by the glob
 *@src - input a glob pattern - a string eg '/**/*.html /*' or '*.html' or, an array eg
```

```
['glob1','glob2']
 *@dest=file.base means, the modified html file will be in the same directory as the src file
being minified
 *often means, the process is just a modification on the existing src file
 *@return returns a stream
 */
var minifyHtml = function(src) {
    return $.src(src)
        .pipe($$.minifyHtml())
        .pipe($.dest(function(file) {
            //file is provided to a dest callback -
            // Refer here https://github.com/gulpjs/gulp/blob/master/docs/API.md#path
            return file.base;
        }));
}
```

Anatomie d'une tâche de gulp

L'anatomie d'une définition de tâche est comme suit:

```
$.task(<taskname> , [dependencies] , <body>);
```

dépendances, est un tableau de tâches qui doivent être terminées avant l'exécution de la tâche en cours. Plus comme forcer une exécution synchrone au lieu de la fonctionnalité asynchrone par défaut.

Ajout de tâches Gulp

Donc, nous avons maintenant

- Une fonction définie ci-dessus pour optimiser les styles
- Une fonction définie ci-dessus pour optimiser les scripts
- Une fonction définie ci-dessus pour optimiser le HTML
- Une fonction pour générer plusieurs images par image ci-dessus

Tout ce que nous devons faire maintenant, c'est de les invoquer en cas de besoin.

Écrivons nos tâches selon la syntaxe définie précédemment

```
/*
* $.task('name','dependency array',function)
results in building a task object as below
task:{
'name':name,
'dep':[array of dependencies],
'fn':function
}
*/


//*@return returns a stream to notify on task completion
$.task('optimizeHtml', function() {
    var src = ['**/*.html', '!{node_modules}/**/*.html'];
    return minifyHtml(src);
});
```

```
//*@return returns a stream to notify on task completion
$.task('optimizeScripts', function() {
    var src = ['js/**/*.js'];
    return optimizeScripts(src);
});

//*@return returns a stream to notify on task completion
$.task('optimizeStyles', function() {
    var src = ['css/**/*.css', 'fonts/google/**/*.css'];
    return optimizeStyles(src);
});

//Take in a callback to ensure notifying the gulp engine, that the task is done
//required since, you are not returning a stream in this task
$.task('generateResponsiveImages', function(callback) {
    var src = ['images/**/*.{jpg,png}'];
    for (var i = widths.length - 1; i >= 0; i--) {
        generateResponsiveImages(src, widths[i]);
    }
    callback();

});
```

Lire Guide complet pour une automatisation du workflow frontal avec Gulpjs -1 of 2 en ligne: https://riptutorial.com/fr/gulp/topic/6342/guide-complet-pour-une-automatisation-du-workflow-frontal-avec-gulpjs--1-of-2

Chapitre 10: Minimiser JS

Syntaxe

- `ext` Objet spécifiant la source de sortie et les extensions de fichier minifiées.
- `source` Chaîne de suffixe des noms de fichiers avec lesquels les fichiers source sortent.
- `min` When string: Chaîne de suffixe des noms de fichiers qui génèrent des fichiers minifiés.
- When Array: expressions d'expressions rationnelles à remplacer par les noms de fichiers d'entrée. Par exemple: [/.(.*)-source.js$/, '$ 1.js']
- `exclude` Ne pas minimiser les fichiers dans les répertoires.
- `noSource` pas le code source dans les répertoires dest.
- `ignoreFiles` Ne pas minifier les fichiers qui correspondent au modèle.
- `mangle` Passe `false` pour ignorer les noms flous.
- `output` Passer un objet si vous souhaitez spécifier des `output options` . Les valeurs par défaut sont optimisées pour une meilleure compression.
- `compress` Passer un objet pour spécifier des `compressor options` personnalisées. Passer false pour ignorer complètement la compression.
- `preserveComments` Une option pratique pour options.output.comments. Par défaut, conserver aucun commentaire.
- `all` conserver tous les commentaires dans les blocs de code
- `some` conservent des commentaires qui commencent par un bang `(!)` ou incluent une directive du compilateur de fermeture `(@preserve, @license, @cc_on)`
- `function` Spécifiez votre propre fonction de préservation des commentaires. Le noeud actuel et le commentaire actuel vous seront transmis et vous devrez retourner `true` ou `false` .

Remarques

Liens utiles pour gulp-minify

Examples

Minimiser JS en utilisant gulp-minify

Tout d'abord, installez `gulp` et `gulp-minify` dans le répertoire du projet localement

```
npm install --save-dev gulp gulp-minify
```

Ajoutez ensuite la tâche `min-js gulpfile.js` à votre `gulpfile.js`

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

gulp.task('min-js', function() {
    return gulp.src('lib/*.js')
        .pipe(minify({
```

```
            ext: {
                min: '.min.js'
            },
            ignoreFiles: ['-min.js']
        }))
        .pipe(gulp.dest('lib'))
});

gulp.task('watch', function(){
  gulp.watch('lib/*.js', ['min-js']);
  // Other watchers
});

gulp.task('default', ['min-js', 'watch']);
```

Cette tâche recherche tous les fichiers js dans le répertoire `lib` , minfy et enregistre dans le répertoire `lib` avec le suffixe `.min.js` . Par exemple, après rapetisser `lib/app.js` fichier sera créé un `lib/app.min.js` fichier

Outre l'exécution en tant que dépendance pour la tâche `'default'` , cette tâche peut être exécutée manuellement en tapant la commande suivante:

```
gulp min-js
```

Lire Minimiser JS en ligne: https://riptutorial.com/fr/gulp/topic/4397/minimiser-js

Chapitre 11: Réduction de CSS

Examples

Utiliser gulp-clean-css et gulp-rename

Tout d'abord, installez `gulp` , `gulp-clean-css` et `gulp-rename` dans le répertoire du projet localy

```
npm install --save-dev gulp gulp-clean-css gulp-rename
```

`minify-css` ajoutez la tâche `minify-css` à votre `gulpfile.js`

```
var gulp = require('gulp');
var cleanCSS = require('gulp-clean-css');
var rename = require('gulp-rename');

gulp.task('minify-css', function() {
    return gulp.src('css/dist/dist.css')
        .pipe(cleanCSS())
        .pipe(rename('dist.min.css'))
        .pipe(gulp.dest('css/dist'));
});

gulp.task('watch', function(){
  gulp.watch('css/dist/**/*.css', ['minify-css']);
  // Other watchers
});

gulp.task('default', ['minify-css', 'watch']);
```

Ici, `.pipe(cleanCSS())` exécute la minification de votre fichier `css/dist/dist.css` et de `.pipe(rename('concat.min.css'))` renomme en `dist.min.css`

Sass and Css - Pré-traitement avec Gulp

Avant de commencer gulp, nous devons installer `node.js` et `npm` . Puis installez gulp-sacc

```
$ npm i gulp-sass --save-dev // i = install
```

Gulp Head

```
var gulp = require('gulp');
// Requires the gulp-sass plugin
var sass = require('gulp-sass');
```

Corps Gulp

```
gulp.task('sass', function(){
  return gulp.src('app/scss/*.scss') // searching for sass files
```

```
    .pipe(sass()) // Converts Sass to CSS with gulp-sass
    .pipe(gulp.dest('app/css')) // destination of the css file
});
```

Montre gulp

```
gulp.task('watch', function(){
  gulp.watch('app/scss/**/*.scss', ['sass']);
  // Other watchers
})
```

Lire Réduction de CSS en ligne: https://riptutorial.com/fr/gulp/topic/4396/reduction-de-css

Chapitre 12: Réduire le code HTML

Examples

Réduire le code HTML à l'aide de gulp-htmlmin

Tout d'abord, installez `gulp` et `gulp-htmlmin` dans le répertoire du projet localement

```
npm install --save-dev gulp gulp-htmlmin
```

Ajoutez ensuite la tâche `minify-html` à votre `gulpfile.js`

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

// Task to minify HTML
gulp.task('minify-html', function() {
return gulp.src('source/*.html')
.pipe(htmlmin())
.pipe(gulp.dest('public/'));
});

gulp.task('watch', function (){
    gulp.watch('source/*.html', ['minify-html']);
// other tasks
});

gulp.task('default', ['minify-html', 'watch']);
```

Cette tâche recherche tous les fichiers dans le répertoire `source` avec une extension `.html` , les minifie, puis affiche les fichiers résultants dans le répertoire `public` .

La tâche dans le code est ajoutée en tant que dépendance pour la tâche `'default'` . Chaque fois que la `default` sera exécutée, `minify-html` sera exécutée avant.

Vous pouvez également appeler la tâche `minify-html` manuellement en exécutant la commande:

```
gulp minify-html
```

Lire Réduire le code HTML en ligne: https://riptutorial.com/fr/gulp/topic/7187/reduire-le-code-html

Chapitre 13: Supprimer des fichiers à l'aide de Gulp

Remarques

Remarque sur l'utilisation du motif de globalisation (`**`):

Le motif de globalisation correspond à tous les `children` **et** au `parent` . Pour éviter cela, nous ajoutons `'!public'` à notre tâche del afin que le répertoire `public` lui-même ne soit pas supprimé

Examples

Supprimer des fichiers en utilisant del

Tout d'abord, installez `gulp` et `del` dans le répertoire du projet localement

```
npm install --save-dev gulp del
```

Puis ajoutez la tâche `clean` à votre `gulpfile.js`

```
var gulp = require('gulp');
var del = require('del');

gulp.task('default', function() {
});

// Task to delete target build folder
gulp.task('clean', function() {
  return del(['public/**', '!public']);
});

gulp.task('default', ['clean']);
```

Cette tâche supprime tous les fichiers du répertoire public

La tâche dans le code est ajoutée en tant que dépendance pour la tâche `'default'` . Chaque fois que la `default` sera exécutée, `clean` s'exécutera avant elle.

Vous pouvez également appeler la tâche de `clean` manuellement en exécutant la commande:

```
gulp clean
```

Lire Supprimer des fichiers à l'aide de Gulp en ligne:
https://riptutorial.com/fr/gulp/topic/7189/supprimer-des-fichiers-a-l-aide-de-gulp

Chapitre 14: Utiliser Browserify

Paramètres

Les options	Détails
transformer	Spécifie un pipeline de fonctions (ou noms de module) à travers lequel le bundle navigué sera exécuté.
déboguer	Activer la prise en charge de la carte source `!gulp.env.production` fonctionnerait bien.
des extensions	Tableau d'extensions que vous souhaitez ignorer dans `require()` appels de `require()` en plus de .js et .json. Ne pas oublier .
ignorer	Tableau de chemins devant être transmis à la fonction ignore de browserify.
résoudre	Fonction de résolution de nom de module personnalisée.
nobuiltins	Supprimez les modules intégrés définis dans `lib/builtins.js` (module browserify). `opts.builtins` doit pas être défini et `opts.nobuiltins` peut être un tableau de chaînes ou simplement une chaîne.

Examples

Utilisation de Browserify avec Javascript Javascript

Installez d'abord gulp et naviguez via `npm i gulp gulp-browserify` . Cela installera browserify dans votre dossier `node_modules` .

gulpfile.js

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

gulp.task('script', function() {
    gulp.src('./src/script.js')
        .pipe(browserify({
            insertGlobals: true
        })
        .pipe(gulp.dest('./build/'));
}
```

Utiliser Browserify avec Coffeescript

Installez d'abord gulp et naviguez via npm i gulp gulp-coffeeify. Cela installera browserify dans votre dossier node_modules.

gulpfile.js

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

gulp.task('script', function() {
    gulp.src('./src/script.coffee')
        .pipe(coffeeify())
        .pipe(gulp.dest('./build/'));
}
```

Lire Utiliser Browserify en ligne: https://riptutorial.com/fr/gulp/topic/6364/utiliser-browserify

Chapitre 15: Utiliser des filtres de fichiers.

Examples

Création d'images, règles JS et CSS (SASS) pour la commande de fichiers

Installer Gulp et ses tâches

```
$ npm install gulp --save-dev
$ npm install gulp-sass --save-dev
$ npm install gulp-uglify --save-dev
$ npm install gulp-imagemin --save-dev
```

Détermination de la structure des dossiers

Dans cette structure, nous utiliserons le dossier app à des fins de développement, tandis que le dossier dist est utilisé pour contenir des fichiers optimisés pour le site de production.

```
|-app
    |-css/
    |-images/
    |-index.html
    |-js/
    |-scss/
|- dist/
|- gulpfile.js
|- node_modules/
|- package.json
```

Prétraitement Gulp

```
// Requires the gulp-sass plugin
var gulp = require('gulp');
    sass = require('gulp-sass');
    uglify = require('gulp-uglify');
    imagemin = require('gulp-imagemin');
```

Tâche Gulp

```
gulp.task('sass', function(){
  return gulp.src('app/scss/**/*.scss') //selection all files in this derectory
    .pipe(sass()) // Using gulp-sass
    .pipe(gulp.dest('dist'))
```

```
});
gulp.task('gulp-uglify', function(){
  return gulp.src('app/js/*.js')
    // Minifies only if it's a JavaScript file
    .pipe(uglify())
    .pipe(gulp.dest('dist'))
});
gulp.task('images', function(){
  return gulp.src('app/images/**/*.+(png|jpg|gif|svg)')
  .pipe(imagemin())
  .pipe(gulp.dest('dist/images'))
});
```

Montre gulp

```
gulp.task('watch', function(){
  gulp.watch('app/js/**/*.js', ['gulp-uglify']);
    gulp.watch('app/scss/**/*.scss', ['sass']);
    gulp.watch('app/images/**/*.*', ['images']);
  // Other watchers
});
gulp.task('build', [`sass`, `gulp-uglify`, `images`], function (){
  console.log('Building files');
});
gulp.task('default', function() {});
```

Lire Utiliser des filtres de fichiers. en ligne: https://riptutorial.com/fr/gulp/topic/7818/utiliser-des-filtres-de-fichiers-

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec gulp	cl3m, Community, fracz, João Silva, naresh, Nhan, Pejman, Pierre-Loup Pagniez, Pyloid, Sender, Tim
2	Afficher les erreurs avec gulp-jslint	jesussegado
3	Chemin Gulp	GYTO
4	Compression sans perte d'image (avec gulp-imagemin)	Siegmeyer
5	Créer un observateur	Muaaz Rafi, Nhan
6	Créer une documentation avec gulp-jsdoc3	Manuel
7	Fichiers concaténés	Andrew Plakhotnyi, fracz, John Strood, maoizm, Mikhail, noob
8	Guide complet d'un flux de travail frontal avec Gulpjs 2 of 2	Schrodinger's cat
9	Guide complet pour une automatisation du workflow frontal avec Gulpjs -1 of 2	Schrodinger's cat
10	Minimiser JS	fracz, GYTO, Mikhail, Mor Paz, Saiful Azad
11	Réduction de CSS	fracz, GYTO, Mikhail, SteveLacy
12	Réduire le code HTML	GYTO, Mor Paz
13	Supprimer des fichiers à l'aide de Gulp	Mor Paz
14	Utiliser Browserify	dome2k
15	Utiliser des filtres de	GYTO

	fichiers.	