



# Workflow Web

**Qu'est-ce que c'est ?**  
**Comment l'optimiser ?**  
**Quels outils peuvent nous aider ?**

# PRÉAMBULE

Dans le quotidien d'un front-end developer, il existe toute une série de choses assez répétitives qu'il doit exécuter pour chaque nouveau projet.

Si, en début de carrière, faire et refaire ces choses n'est pas un problème. À la longue, cela peut devenir lassant et c'est surtout complètement contre-productif.

Un workflow Web, c'est en quelque sorte un « ensemble d'étapes » par lesquelles nous allons devoir passer pour chaque projet afin d'arriver au résultat final.

Une fois qu'on a compris pourquoi on doit faire certaines choses, il est bon de les automatiser.

Et c'est là qu'une série d'outils est à notre disposition afin de faciliter notre vie de front-end developer.

Ces outils s'appellent l'invite de commande, Gulp (ou Grunt) et toutes les dépendances qui accompagnent ces outils.

L'automatisation de certaines tâches de votre Workflow Web va vous faire gagner un temps précieux et vous permettre surtout de vous concentrer uniquement sur le cœur-même de vos projets.

Voyons donc ensemble ce qui compose un Workflow Web et comment va-t-on pouvoir l'optimiser grâce, notamment, à Gulp.



Gulp

# TABLE DES MATIÈRES

<b>PRÉAMBULE</b>	<b>2</b>
<b>WORKFLOW WEB</b>	<b>4</b>
• Définition .....	4
• Inception : un workflow dans notre workflow Web.....	5
• La solution s'appelle Gulp .....	5
• Pré-requis pour utiliser Gulp .....	6
Node.js	6
Créer son environnement de travail	7
package.json	9
gulpfile.js	10
Déclaration des plugins	10
Les tâches	11
L'exécution des tâches	12
• Quelques plugins utiles .....	13
Gulp-sass	13
Gulp-autoprefixer	13
Gulp-clean-css	14
Gulp-plumber & Gulp-notify	14
Gulp-uglify	15
Gulp-rename	16
Browser-sync	16
Mais encore ...	17
• Conclusion .....	17

# WORKFLOW WEB

## ● Définition

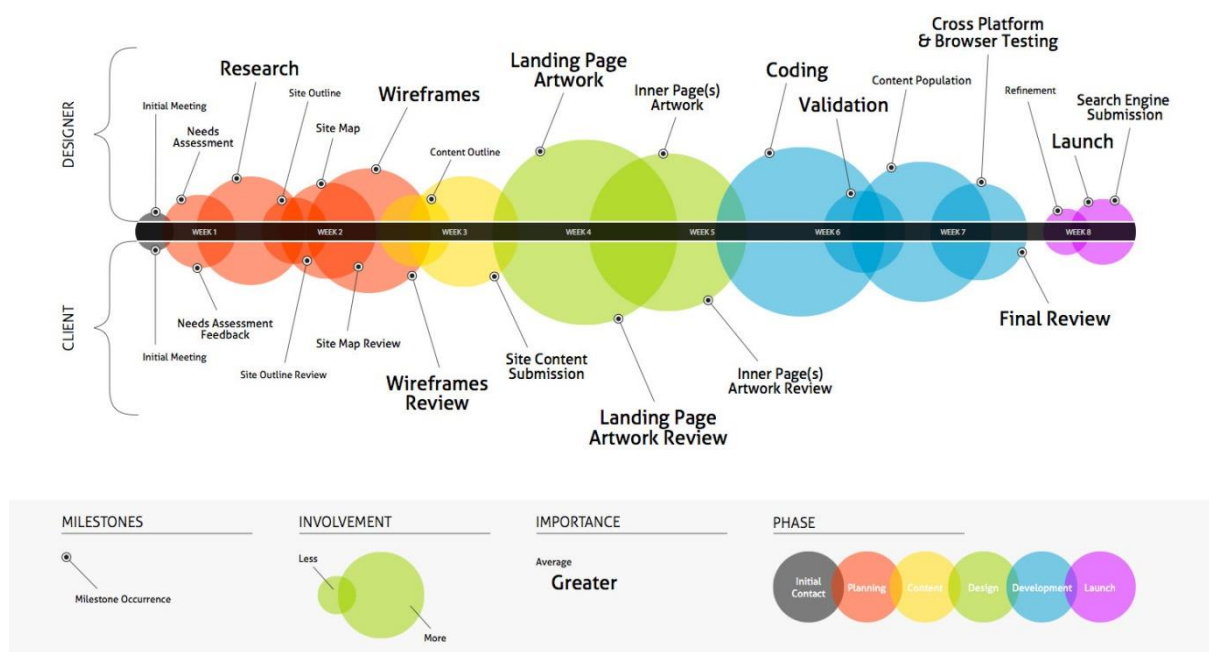
Un workflow, par définition, désigne un ensemble d'étapes par lesquelles on va devoir passer pour arriver au résultat final.

Pour le Web, il y a toute une série d'étapes qui constituent la création d'un site Web et même si certaines ne sont pas de notre ressort, il est bon de les rappeler rapidement :

1. Recherche
2. Site map
3. Wireframes
4. Définition du contenu (dans les grandes lignes)
5. Création graphique de la page d'accueil
6. Création graphique des pages annexes
7. Codage de toutes ces pages, validation du code et testing cross-browser / plateforme
8. Injection du contenu
9. Mise en ligne du site
10. Soumission aux moteurs de recherche

### A Web Site Designed

MILESTONES, INVOLVEMENT, IMPORTANCE & TIMELINE



Dans votre métier, vous allez être amenés à parcourir ces différentes étapes pour produire le site Web que votre client vous a demandé.

Nous allons mettre principalement l'accent sur certaines étapes de ce Workflow : les étapes 7 à 9. Ce sont celles qui concernent principalement le codage du site Web.

## ● Inception : un workflow dans notre workflow Web

En plus d'avoir un workflow Web général à la création de votre site Web, vous allez aussi mettre en place, avec un peu d'expérience, tout un certain nombre de choses répétitives lors de la création d'une page Web. Ces étapes sont souvent rébarbatives et pas très amusantes :

1. Création des fichiers / dossiers nécessaires à l'intégration complète de votre projet (html, css, js)
2. Initialisation et installation de certaines bibliothèques js / css
3. Déclaration de certaines règles récurrentes (box-sizing : border-box, et ce genre de choses qu'on a tendance à mettre partout, tout le temps)
4. Lorsque vous utilisez du CSS3, placement des préfixes des browsers pour la compatibilité avec tous les navigateurs et leurs différentes versions
5. Rafraîchissement de votre page à chaque demi-modification que vous effectuez, pour voir ce que ça fait
6. Minification de votre CSS
7. Minification de votre JS
8. Compression de vos images
9. Compilation de vos fichiers SASS vers CSS
10. Mise en ligne des changements effectués sur le serveur distant
11. Etc.

Toutes ces tâches, vous allez devoir les effectuer une à une, séparément et un nombre incalculable de fois au sein d'un même projet mais aussi de façon répétitive lors de chaque projet.

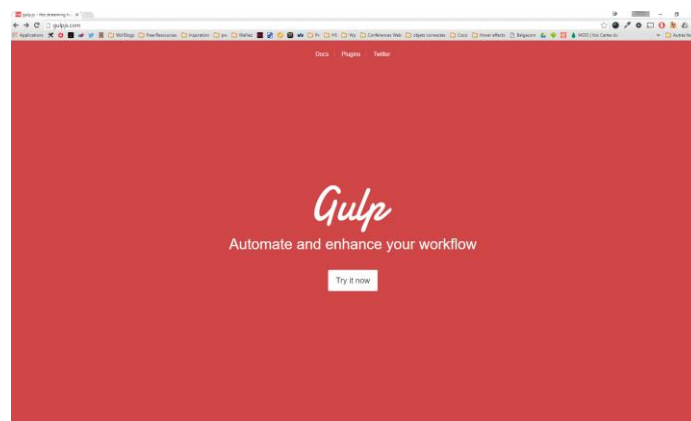
C'est une perte de temps considérable. Votre temps est précieux !

Et pour vous aider, il va falloir bien vous entourer.

## ● La solution s'appelle Gulp

Gulp est un outil gratuit (!) qui a été créé pour vous décharger de toute une série de tâches rébarbatives dans votre quotidien.

Son slogan n'est autre que « Automate and enhance your workflow » (automatisez et améliorez votre Workflow).



Gulp regroupe plus de 2300 « plugins » qui, chacun, peuvent automatiser une tâche.

Parmi les tâches les plus communes et les plus utilisées, nous pouvons citer :

- Compilation automatique de vos fichiers SASS en CSS
- Automatisation des préfixes sur les propriétés CSS quand nécessaire
- Minification du CSS
- Minification du JS
- Optimisation du poids des images
- Renommage de fichiers
- Rafraîchissement automatique des browsers à chaque sauvegarde
- Notification des erreurs de CSS ou de JS dans l'invite de commande
- Envoi en FTP
- Suppression des règles CSS non utilisées dans notre projet
- Liaison avec Git
- Etc.

Chaque plugin (que vous allez devoir installer en ligne de commande) remplit une fonction qu'il vous faudra paramétrer pour qu'elle s'adapte à votre façon de travailler.

## ● Pré-requis pour utiliser Gulp

Pour pouvoir commencer à utiliser Gulp, vous n'aurez besoin que de node.js.



Tout le reste se passe dans l'invite de commande (oui vous allez pouvoir faire comme dans les films et coder dans une interface sur fond noir complètement rébarbative – c'est ultime ^^).

## Node.js

Un petit tour sur le site officiel de node pour aller installer la version de node qui correspond à votre OS et la version de votre OS et le tour est joué !



Une fois que node.js est installé, vous ouvrez l'invite de commande et vous tapez :

```
npm install gulp -g
```

Cette première commande n'est à effectuer qu'une seule et unique fois sur votre machine, pour tous vos projets avec Gulp. Le `-g` veut dire qu'on l'installe de façon globale, sur toute votre machine tandis que la commande `npm` est la marque de Node et `install` est une commande de node.

### Attention

Si vous êtes sur Mac ou Linux, vous aurez sans doute besoin de faire précéder cette syntaxe d'un `sudo` afin de disposer des droits suffisants, ce qui donnera :

```
sudo npm install gulp -g
```

Les droits d'admin sur Mac peuvent compliquer certaines choses dans les démarches qui suivent, soyez prudents.

Et voilà ! Gulp est installé sur votre machine ! C'est cool non ?

Bon en fait, pas tant que ça parce que dans l'état Gulp ne sert pas à grand-chose tant qu'on n'a pas installé de « dépendance » (plugin) et surtout, pour fonctionner correctement vous avez encore besoin de 2 fichiers à la racine de votre projet :

- `Package.json` : contient la liste des plugins Gulp nécessaires à vos tâches
- `Gulpfile.js` : contient la liste des tâches à réaliser

## Créer son environnement de travail

Avant même de créer les 2 fichiers ci-dessus, créons ensemble une structure de dossiers et de fichiers que l'on va pouvoir réutiliser pour tous nos projets.

Cette structure pourrait comprendre, dans le dossier au nom de notre projet, un dossier `src` et un dossier `dist` (source et distribution).

Dans `src` nous allons mettre tous nos fichiers de travail écrits en SASS, nos fichiers js non-minifiés, nos images non compressées, etc.

Alors que `dist` sera votre dossier de production. C'est-à-dire qu'à peu de choses près, vous pourrez prendre et déposer ce dossier sur votre serveur distant et tout devrait fonctionner. C'est votre site fini.

Vous pouvez déjà créer des fichiers `.html` vierges et inclure les fichiers habituels que vous liez à vos projets, si vous en avez (travailler en CDN est une meilleure solution).

Pour les fichiers Sass, une pratique courante est de tout « découper » en plein de petits fichiers.

Mais pourquoi cela ?

C'est simple !

D'abord pour pouvoir facilement s'y retrouver et savoir exactement quel fichier vous allez devoir éditer pour effectuer une modification.

Ensuite, ce côté « modulaire » est aussi une excellente façon de pouvoir reprendre une feuille sass dans un projet et de l'utiliser dans un autre projet. Je pense principalement aux Mixins, à des feuilles Sass pour vos boutons, pour le reset, etc.

Au final, tout sera de toute façon réuni à l'intérieur d'un seul et même fichier.

Voici un exemple d'architecture de fichiers Sass que l'on pourrait utiliser (source : <http://www.sitepoint.com/architecture-sass-project/>), vous n'êtes évidemment pas obligés de vous y tenir, mais c'est une bonne base de réflexion pour votre propre architecture.

```
sass/
|
|- base/
|   |- _reset.scss           # Reset/normalize
|   |- _typography.scss      # Typography rules
|   ...                      # Etc...
|
|- components/
|   |- _buttons.scss         # Buttons
|   |- _carousel.scss        # Carousel
|   |- _cover.scss           # Cover
|   |- _dropdown.scss        # Dropdown
|   |- _navigation.scss      # Navigation
|   ...                      # Etc...
|
|- helpers/
|   |- _variables.scss       # Sass Variables
|   |- _functions.scss       # Sass Functions
|   |- _mixins.scss          # Sass Mixins
|   |- _helpers.scss         # Class & placeholders helpers
|   ...                      # Etc...
|
|- layout/
|   |- _grid.scss            # Grid system
|   |- _header.scss          # Header
|   |- _footer.scss          # Footer
|   |- _sidebar.scss         # Sidebar
|   |- _forms.scss           # Forms
|   ...                      # Etc...
|
|- pages/
|   |- _home.scss            # Home specific styles
|   |- _contact.scss         # Contact specific styles
|   ...                      # Etc...
|
|- themes/
|   |- _theme.scss           # Default theme
|   |- _admin.scss           # Admin theme
|   ...                      # Etc...
|
|- vendors/
|   |- _bootstrap.scss       # Bootstrap
|   |- _jquery-ui.scss       # jQuery UI
|   ...                      # Etc...
|
|- main.scss                 # primary Sass file
```

Un brin d'explications sur cette architecture de fichiers.

Dans `base/`, vous trouverez des fichiers que vous modifierez rarement comme votre reset (ou normalize), vos appels à vos typo, etc.

Dans `components/`, vous trouverez un ensemble de petits modules indépendants du genre : buttons, carousel, navigation, dropdown, etc.



Dans `helpers/`, vous trouverez des règles qui ne sont là que pour vous aider, c'est-à-dire qu'on ne définit rien de la structure de notre site ici. On met en place des variables, des fonctions et des mixins qu'on va pouvoir utiliser durant toute la durée de notre projet.

Dans `layout/`, vous trouverez votre mise en page générale, alors que components concerne des petits modules, layout, lui, s'occupe des grandes dispositions d'éléments sur vos pages (grille de mise en page, header, footer, sidebar, formulaires, etc.)

Dans `pages/`, vous trouverez les styles spécifiques à certaines pages de votre site, on en a dans tous nos projets.

Si vous travaillez sur un très grand site, alors `themes/` pourrait être utile. Dans les autres cas vous ne l'utiliserez pas.

Dans `vendors/`, vous trouverez les feuilles de styles externes qui sont issues des outils que vous allez utiliser. Par exemple on peut retrouver les fichiers Sass pour Bootstrap, jquery-ui, fontawesome, etc.

Tout cela n'a qu'un but : gagner du temps. Dès lors il est logique de créer un dossier de travail « de base » pour tous vos projets Web.

## package.json

Pour générer le premier (package.json) il vous suffit de lancer la commande suivante à la racine de votre projet :

```
npm init
```

Toute une série d'informations vous seront alors demandées concernant votre projet :

- Son nom
- La licence
- Etc.

Il vous suffit de répondre aux questions et de valider vos choix avec la touche « Enter ».

L'étape qui suit est de déclarer et installer un par un chaque plugin qui sera nécessaire pour votre projet, en commençant par Gulp lui-même :

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

Ensuite vous installez tous les plugins les uns après les autres de la même manière, par exemple pour sass :

```
npm install gulp-sass --save-dev
```

Mais le plus simple est probablement d'aller rechercher un package.json déjà existant (d'un projet précédent) et de le coller dans votre nouveau projet. Ensuite si vous lancez simplement la ligne suivante, toutes les dépendances de votre ancien projet seront installées :

```
npm install
```

En réalité, npm install va chercher tous les plugins listés dans votre fichier package.json et les installe.

Un fichier package.json assez classique ressemble à ceci :

```
{
  "name": "projet",
  "version": "0.0.1",
  "devDependencies": {
    "gulp": "latest",
    "gulp-load-plugins": "latest",
    "gulp-rename": "latest",
    "gulp-cssso": "latest",
    "gulp-less": "latest",
    "gulp-csscomb": "latest",
    "gulp-cssbeautify": "latest",
    "gulp-autoprefixer": "latest"
  }
}
```

L'ensemble des plugins (ou dépendances) vont être installées dans un dossier node\_modules qui npm install aura généré. Et dans ce dossier node\_modules vous retrouverez un dossier par plugin.

#### Attention

Supprimer le dossier node\_modules sous Windows est compliqué à cause de l'innombrable nombre de fichiers qui sont contenus dans ces dossiers. Il vous faudra installer RimRaf pour y parvenir en invite de commande.

C'est facilité pour les utilisateurs Mac.

## gulpfile.js

Ce deuxième fichier est capital puisqu'il s'occupe de gérer les tâches à réaliser quand la « moulinette » de Gulp est lancée (pour le moment nous ne l'avons pas encore lancée, nous avons simplement installé Gulp et ses dépendances).

Notre fichier gulpfile.js est composé de 3 parties : une partie de **déclarations** (variables et plugins), les **tâches** à proprement parler et **l'exécution de ces tâches** et dans quel contexte cette exécution va avoir lieu.

## Déclaration des plugins

C'est la première partie de votre fichier, on va y retrouver la déclaration de tous nos plugins.

On va attribuer une variable pour chaque plugin de cette façon :

```
var gulp = require('gulp');
var sass = require('gulp-sass');
var autoprefixer = require('gulp-autoprefixer');
```

Vous aurez donc une ligne en plus pour chaque plugin que vous allez installer avec `npm install`.

Dans la partie « require », vous devez exactement indiquer le nom du plugin tel que vous l'avez installé. Par exemple 'gulp-sass' si vous avez installé `npm install gulp-sass --save-dev`.

Pour le nom des variables, en revanche, vous êtes totalement libres. Vous devrez juste utiliser la variable que vous aurez déclarée lorsque vous allez rédiger la tâche qui est en rapport avec le plugin en question.

## Les tâches

C'est un peu le cœur de notre gulpfile.

Dans un premier temps nous avons déclaré les plugins et nous avons stocké ce qu'ils contiennent dans une variable. Nous allons maintenant mettre ces variables en œuvre à travers différentes tâches.

On déclare une tâche de cette façon dans gulp :

```
gulp.task('nomdelatache', function () {  
});
```

Quand vous créez une tâche avec gulp, vous devez lui donner un nom (différent du nom de la variable que vous avez choisi auparavant). Par exemple, pour faire la conversion de notre Sass en Css nous pourrions utiliser « sassification » comme ceci :

```
gulp.task('sassification', function () {  
});
```

Reste que, avec juste ça, il ne se passe rien.

Il va falloir indiquer où la moulinette de Sass doit aller chercher mes fichiers Sass et où il doit aller les sauvegarder.

Si vous ne savez pas comment vous y prendre pour rédiger une tâche, sachez que pour chaque tâche, il existe une page d'aide qui lui correspond. Par exemple, pour gulp-sass, vous retrouverez sur cette fameuse page des informations capitales.

Comment installer gulp sass (source : <https://www.npmjs.com/package/gulp-sass>) :

```
npm install gulp-sass --save-dev
```

Quelle utilisation de base je peux faire de ce plugin :

```
gulp.task('sass', function () {  
  return gulp.src('./sass/**/*.scss')  
    .pipe(sass().on('error', sass.logError));  
    .pipe(gulp.dest('./css'));  
});
```

Imagions que cette portion de code ci-dessus vous fasse peur (ce que je pourrais comprendre). Nous allons la décortiquer pour se rendre compte qu'en fait tout est compréhensible et demande certaines petites adaptations pour correspondre à la structure de nos dossiers sur notre machine.

Je vous épargne la première et la dernière ligne qui sont l'ouverture de la tâche Sass et sa fermeture.

```
return gulp.src('./src/sass/*.scss')
```

La ligne ci-dessus indique où Sass doit aller chercher mes fichiers .scss. Dans ce cas-ci, par rapport à où se trouve mon fichier gulpfile, dans un dossier src puis dans un dossier sass et là prendre tous les fichiers qui ont pour extension \*.scss.

Deux instructions « .pipe » interviennent ensuite, il s'agit de lignes où il y a une « action » dans notre gulpfile.

La première action « sass » sert à prendre tout ce qui a été pris dans le Return de la ligne juste avant et à la convertir du langage Sass au langage Css :

```
.pipe(sass().on('error', sass.logError));
```

En cas d'erreur dans le fichier Sass, une erreur apparaîtra dans la console et Gulp va s'arrêter.

Ensuite vient une instruction gulp-dest qui sauvegarde le résultat de la traduction de l'action précédente à un endroit précis :

```
.pipe(gulp.dest('./dist/css'));
```

## L'exécution des tâches

Vous venez de créer votre première tâche mais vous ne l'avez pas encore lancée donc pour le moment il ne se passe juste rien.

Deux méthodes pour lancer une tâche.

Soit manuellement dans la console en écrivant :

```
gulp sassification
```

En écrivant ça, vous lancez expressément la tâche que vous venez de créer mais aucune autre. Et surtout, gros point noir, vous allez devoir le faire à chaque fois que vous avez effectué un changement dans votre fichier Sass et ça, ça va juste à contre-courant de l'esprit même de Gulp qui veut vous faire gagner du temps.

Il va donc falloir écrire une tâche de surveillance qui va lancer automatiquement cette tâche (et potentiellement aussi toutes les autres) quand vous changez quelque chose dans un fichier Sass (dans ce cas-ci).

```
gulp.task('watch', function () {  
  gulp.watch('./src/sass/**/*.scss', ['sassification']);  
});
```

Cette tâche, nous l'appellerons « watch » et ce qu'elle va faire c'est que, à chaque changement d'un fichier .scss se trouvant dans le dossier .src/sass/ ou dans n'importe quel sous-dossier (/\*\*/) il va relancer la tâche « sassification ». C'est-à-dire qu'il va recompiler le fichier Sass vers Css.

Finalement, nous devons attribuer la tâche « watch » au comportement de gulp par défaut de cette manière :

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

Dans la console, lancez maintenant simple gulp comme ceci : `gulp` (+enter). Gulp va entrer en mode « surveillance » puisque vous lui avez demandé que ce soit son comportement par défaut.

Magique non ? Plus besoin de relancer la tâche ! Elle s'exécute automatiquement à chaque sauvegarde d'un de vos fichiers .scss. (c'est trop stylé ^^).

## • Quelques plugins utiles

### Gulp-sass

Nous en avons parlé dans le chapitre précédent, pas besoin de revenir sur ce plugin qui sert donc bien de moulinette qui transforme votre .scss en .css et l'enregistre à l'endroit désiré.

### Gulp-autoprefixer

Gulp-autoprefixer est là pour vous épargner d'ouvrir le site caniuse.com pour voir si une propriété CSS3 récente est supportée par tous les navigateurs. Il préfixe les navigateurs qui en ont besoin avec le préfixe adéquat pour chaque propriété CSS. Vous pouvez toujours avoir accès aux options sur la page du plugin (<https://www.npmjs.com/package/gulp-autoprefixer/>).

#### Installation

```
npm install --save-dev gulp-autoprefixer
```

#### Utilisation

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

gulp.task('sassification', function () {
  return gulp.src('./src/sass/*.scss')
    .pipe(sass())
    .pipe(autoprefixer({
      browsers: ['last 2 versions'],
      cascade: false
    }))
    .pipe(gulp.dest('./dist/css'));
});
```

En **orange**, ce qui change dans votre tâche « sassification ».

Dans un premier temps, on déclare la nouvelle variable « autoprefixer » puis on injecte la tâche d'autoprefixage avec .pipe au sein même de la tâche « sassification » pour que tout soit fait à chaque fois que la tâche « sassification » est appelée.

On redistribue toujours le résultat au même endroit, rien ne change à ce niveau évidemment.

## Gulp-clean-css

Gulp-clean-css vous permet de nettoyer votre code CSS et de le minifier. Si vous ne précisez pas d'option, il va juste le minifier et c'est déjà super. Vous pouvez toujours avoir accès aux options sur la page du plugin (<https://www.npmjs.com/package/gulp-clean-css/>).

### Installation

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

### Utilisation

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

gulp.task('sassification', function () {
  return gulp.src('./src/sass/*.scss')
    .pipe(sass())
    .pipe(autoprefixer({
      browsers: ['last 2 versions'],
      cascade: false
    })))
    .pipe(cleancss())
    .pipe(gulp.dest('./dist/css'));
});
```

En **orange**, ce qui change dans votre tâche « sassification ».

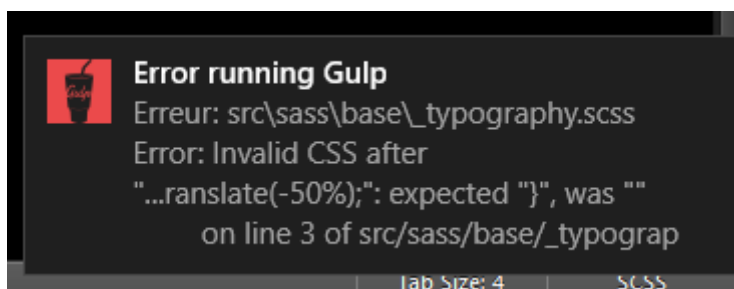
On minifie le .css une fois qu'il a été auto-préfixé et avant d'aller le sauvegarder dans sa destination finale.

## Gulp-plumber & Gulp-notify

Avec Gulp, quand vous faites une erreur dans un fichier .scss et que la moulinette essaye de la compiler en .css, ça arrête la tâche de surveillance et vous devez la relancer, et ça c'est pas glamour.

Alors pour contrer ce problème-là, on installe 2 plugins.

Gulp-plumber est là pour ne plus arrêter la tâche quand il y a une erreur de code et vous dire où se trouve l'erreur en question tandis que gulp-notify est là pour rendre un peu « jolies » les erreurs qui vont être envoyées du genre :



Bon ça reste assez effrayant mais c'est quand même bien d'avoir cette petite boîte de dialogique qui apparaît (sous windows ici) puisque ça vous évite de garder votre console ouverte pour voir si vous avez fait une erreur.

Néanmoins dans la console, on a aussi un message :

```
[15:05:18] gulp-notify: [Error running Gulp] Erreur: src\sass\base\_typography.scss
Error: Invalid CSS after "...ranslate(-50%);": expected "}", was "" as it is passed through
on line 3 of src/sass/base/_typography.scss
>> transform:translate(-50%);
-----^
```

En pratique, voyons ce qu'il faut faire pour avoir droit à ces notifications.

### Installation

```
npm install --save-dev gulp-autoprefixer gulp-notify
```

### Le saviez-vous ?

On peut installer 2 plugins d'un coup en les séparant par un espace ! C'est trop cool !

### Utilisation

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

gulp.task('sassification', function () {
  return gulp.src('./src/sass/*.scss')
    .pipe(plumber({errorHandler: notify.onError("Erreur: <%= error.message %>")}))
    .pipe(sass())
    .pipe(autoprefixer({
      browsers: ['last 2 versions'],
      cascade: false
    }))
    .pipe(gulp.dest('./dist/css'));
});
```

En **orange**, ce qui change dans votre tâche « sassification ».

## Gulp-uglify

Gulp-uglify agit comme gulp-clean-css, c'est-à-dire qu'il s'agit d'un outil de minification mais cette fois-ci réservé à vos fichiers javascript (.js).

### Installation

```
npm install --save-dev gulp-uglify
```

### Utilisation

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

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

Cette fois, il va falloir créer une nouvelle tâche puisqu'on est plus dans un fichier qui à attrait au CSS mais bien au javascript. Cette tâche fonctionne comme celle pour Sass, c'est-à-dire qu'on va chercher à un endroit nos fichiers .js, on les minifie avec uglify et on les sauve ailleurs.

Ensuite il faut appeler cette nouvelle tâche dans notre tâche par défaut :

```
gulp.task('watch', function () {
  gulp.watch('./src/sass/**/*.scss', ['sassification']);
  gulp.watch('./src/js/**/*.js', ['uglification']);
});
```

## Gulp-rename

Gulp-rename sert simplement à renommer un fichier. C'est utile pour la minification de vos fichiers puisqu'une convention de nommage est en place pour ce genre de fichiers. En effet, les versions minifiées se terminent toujours par .min.js ou .min.css. C'est simplement à ça que va servir ce plugin : à transformer le nom de votre fichier.

### Installation

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

### Utilisation

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

gulp.task('uglification', function() {
  return gulp.src('./src/js/*.js')
    .pipe(rename(function(path) {
      path.basename = path.basename.replace(".src", ".min");
    }))
    .pipe(uglify())
    .pipe(gulp.dest('./dist/js'));
});
```

Dans ce cas-ci, nous n'allons agir que sur notre tâche « uglification » et nous allons lui dire d'aller chercher tous les fichiers qui comportent « .src » et de remplacer cette chaîne de caractères par « .min » et le tour est joué.

## Browser-sync

Browser-sync est probablement le plugin qui plait le plus aux front-end developers. Il est très utile et offre un confort non négligeable en production. Ce que fait Browser-sync est simple : il offre un refresh automatique de votre browser à chaque fois que vous enregistrez un fichier. Plus besoin d'aller faire des « refresh » à la main dans vos browsers.

Une autre fonctionnalité super-intéressante de Browser-sync c'est que vous allez pouvoir consulter le résultat de votre travail sur plusieurs devices en même temps et même sur les devices mobiles ! C'est le côté « magique » de Browser-sync. Quand vous ferez défiler votre page sur un device, elle défilera partout ailleurs de la même façon. Incroyable ! 😊

### Installation

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



## Utilisation

```
var browserSync = require('browser-sync');

gulp.task('browser-sync', function() {
  browserSync.init({
    server: {
      baseDir: "./dist/"
    }
  });
});
```

Nous créons une nouvelle tâche dans laquelle nous allons simplement préciser « où » browser-sync doit se synchroniser. Ici dans le répertoire ./dist/. C'est-à-dire que c'est exactement cet endroit dans lequel browser-sync va se lancer par défaut.

Ensuite, à chaque changement sur un fichier .js, .css ou .html nous aimerions que la page se rafraîchisse. Pour y parvenir nous allons éditer notre tâche d'écoute de cette façon :

```
gulp.task('watch', ['browser-sync'], function () {
  gulp.watch('./src/sass/**/*.scss', ['sassification']);
  gulp.watch('./src/js/**/*.js', ['uglifyfication']);
  gulp.watch('./dist/*.html').on('change', browserSync.reload);
  gulp.watch('./dist/css/*.css').on('change', browserSync.reload);
  gulp.watch('./dist/js/*.js').on('change', browserSync.reload);
});
```

On dit simplement à notre tâche de relancer les browsers quand elle perçoit un changement dans un fichier .html, .css ou .js dans les répertoires indiqués.

## Mais encore ...

Il existe plus de 2000 plugins, nous n'allons pas les voir tous les uns après les autres mais il en reste pas mal qui sont super-intéressants. En voici encore une liste non-exhaustive :

- Gulp-htmlmin : minifie l'html
- Gulp-csslint : valide votre code CSS au W3C
- Gulp-imagemin : compression d'image
- Gulp-git : pour utiliser des commandes Git
- Etc.

## ● Conclusion

Gulp est un formidable outil de production. Il vaut la peine de prendre un peu de temps pour comprendre comment il fonctionne. Une fois que vous aurez pris cet outil en main, vous gagnerez un temps précieux pour tous vos projets.

Investissez votre temps sur des choses plus essentielles et sur d'autres apprentissages puisque Gulp est devenu officiellement votre meilleur ami en prenant en charge toutes ces tâches rébarbatives.

We love Gulp ♥