webdesign, javascript, node.js, npm, grunt, modernizr, linux

Késako ?

Lorsque l'on met le doigt dans la création d'un site Web, on finit forcément par entendre parler de ces outils…

Du point de vue d'un béotien comme moi, il s'agit d'une plateforme Javascript permettant d'utiliser un nombre impressionnant de packages globalement liés au développement (mais pas que).

Je ne vais pas me risquer à entrer dans le détail car je ne maîtrise pas du tout (loin de là)… 

La première fois que je suis tombé sur `npm`, j'ai regardé comment procéder, j'ai lu quelques pages, je suis tombé sur `Node.js`, j'ai encore lu et je me suis dit *« Heu… OK ça à l'air super mais on peut faire sans, donc… ».*

Sauf que… Ben sauf que dans certains cas, on ne peut pas vraiment faire sans. 

Dans mon cas c'est d'abord le désir d'utiliser Modernizr qui m'a obligé à chercher plus : depuis sa dernière évolution majeure, `Modernizr` n'existe plus en tant que package complet prêt à télécharger (la raison est simple : il grossit irrémédiablement et il est devenu nécessaire de ne charger que les parties nécessaires), il faut sélectionner les éléments que l'on souhaite inclure. Le problème est le suivant : si on ne pense pas à inclure un test donné et qu'il faut refaire une sélection, le seul moyen de procéder sans refaire sa sélection et donc risquer d'oublier des éléments précédemment sélectionnés est d'utiliser `npm`.

Tous les outils dont il est question sur cette page sont développés en Javascript, un langage qui n'est pas capable de dialoguer directement avec un OS et pour lequel il faut utiliser un moteur Javascript. Par ailleurs, plutôt que de développer à chaque fois un logiciel complet, les développeurs créent des packages qui utilisent les fonctions de packages d'autres développeurs et ainsi de suite. C'est ce qui explique que pour utiliser tous ces outils, il faut créer une plateforme Javascript, j'ai même vu plusieurs fois le terme “écosystème”.

A chaque niveau de cet écosystème, il faut choisir quel outil utiliser… Mais, lorsque l'on n'a pas les connaissances nécessaire pour décider, il suffit de se laisser guider par les piste laissées par chaque développeur pour utiliser tel ou tel package.

Voici donc les différents éléments de la plateforme Javascript que j'ai mise en place: Node.js, npm et GRUNT.

L'installation

Node.js et npm

Comme il est ici question d'utiliser des outils “à la pointe” et que Debian n'est pas le champion des

versions les plus récentes des packages, il vaut mieux tout compiler. Pour l'un comme pour l'autre, l'installation est plutôt *velue*. 😳
Mais il existe un script *presque* bien fait pour installer tout çà. Je dis seulement “presque bien fait” car il gère malheureusement mal certaines dépendances (il y en a sans doute d'autres dont certains fonctionnent peut-être mieux mais bon). Ce qui est pénible, c'est que certaines étapes du script sont terriblement longues (la compilation de `Node.js` prends vraiment un temps fou… plus d'une heure sur mon système), au point qu'on ne sait pas trop s'il est planté, attends une action ou travaille…

Pour commencer, il faut installer certains paquets : le script part du principe que `make` est installé par exemple et par ailleurs il est supposé gérer seul l'installation de g++ mais cela n'a pas fonctionné dans mon cas.

<cli>root@muffin:~# aptitude install g++ make git</cli>

Et il faut aussi préparer la variable *PATH* pour `Node.js` et `npm` (le script suggère la modification à la fin alors que c'est un pré-requis pour compiler `npm` 😕) en ajoutant à ce fichier :

<cli>root@muffin:/tmp# vi ~/.bashrc</cli>

Les deux lignes suivantes :

```
export PATH="$PATH:/opt/node/bin"
export NODE_PATH="/opt/node:/opt/node/lib/node_modules"
```

On peut ensuite utiliser le script :

<cli>root@muffin:~# cd /tmp root@muffin:/tmp# wget https://raw.github.com/nicolargo/nodeautoinstall/master/nodeautoinstall.py root@muffin:/tmp# python ./nodeautoinstall.py -d Install pre-requisites <color chartreuse>[OK]</color> Download NodeJS <color chartreuse>[OK]</color> Install NodeJS (please wait…) <color chartreuse>[OK]</color> Install NPM (please wait again and longer…) <color chartreuse>[OK]</color> Optionnaly, add the following lines to your .profile file: # — NodeJS export PATH=$PATH:/opt/node/bin export NODE_PATH=/opt/node:/opt/node/lib/node_modules # — </cli>

Là où c'est écrit <color gold>[Warning]</color> au lieu de <color chartreuse>[OK]</color>, il y a eu un problème et il faut aller regarder le fichier de log (`vi /tmp/nodeautoinstall.log`) pour trouver la commande qui n'a pas fonctionné. Il ne reste qu'à comprendre pourquoi puis, une fois le problème résolu, on peut relancer le script.

GRUNT

Grunt est un lanceur de tâches se basant sur `npm`. Il n'est pas nécessaire de l'installer par exemple pour utiliser le module permettant de télécharger Modernizer, il est par contre indispensable pour les outils comme `postcss`.

L'installation est bien plus simple puisqu'elle est gérée directement par `npm` :

<cli>root@muffin:~# npm install -g grunt-cli /opt/node/bin/grunt →

/opt/node/lib/node_modules/grunt-cli/bin/grunt /opt/node/lib └─┬ grunt-cli@1.2.0

```
├─┬ findup-sync@0.3.0
│ └─┬ glob@5.0.15
│   ├─┬ inflight@1.0.5
│   │ └── wrappy@1.0.2
│   ├── inherits@2.0.1
│   ├─┬ minimatch@3.0.0
│   │ └─┬ brace-expansion@1.1.4
│   │   ├── balanced-match@0.4.1
│   │   └── concat-map@0.0.1
│   ├── once@1.3.3
│   └── path-is-absolute@1.0.0
├── grunt-known-options@1.1.0
├─┬ nopt@3.0.6
│ └── abbrev@1.0.7
└── resolve@1.1.7</cli>
```

<cli>root@muffin:~# grunt –version grunt-cli v1.2.0</cli>

Modernizr

Une fois npm installé, on peut lui ajouter des modules, par exemple celui de Modernizr :
<cli>root@muffin:/tmp# npm install -g modernizr /opt/node/bin/modernizr → /opt/node/lib/node_modules/modernizr/bin/modernizr /opt/node/lib └─┬ modernizr@3.3.1

```
├─┬ doctrine@1.1.0
│ ├── esutils@1.1.6
│ └── isarray@0.0.1
├── file@0.2.2
├── find-parent-dir@0.3.0
├── lodash@4.0.0
├── marked@0.3.5
├─┬ mkdirp@0.5.1
│ └── minimist@0.0.8
├── requirejs@2.1.22
└─┬ yargs@3.31.0
  ├── camelcase@2.1.0
  ├─┬ cliui@3.1.0
  │ ├─┬ strip-ansi@3.0.0
  │ │ └── ansi-regex@2.0.0
  │ └── wrap-ansi@1.0.0
  ├─┬ decamelize@1.1.2
  │ └── escape-string-regexp@1.0.4
  ├─┬ os-locale@1.4.0
  │ └─┬ lcid@1.0.0
  │   └── invert-kv@1.0.0
  ├─┬ string-width@1.0.1
```

```
  │ ├─┬ code-point-at@1.0.0
  │ │ └── number-is-nan@1.0.0
  │ └── is-fullwidth-code-point@1.0.0
  ├── window-size@0.1.4
  └── y18n@3.2.0
```

</cli>

Et pour aller au bout de l'exemple...

Lors d'un téléchargement manuel de `Modernizr`, il propose le fichier `modernizr-config.json` et c'est lui qui contient la liste des éléments sélectionnés. Pour l'utiliser, il suffit de le placer dans un dossier quelconque (disons `~/modernizr` pour l'exemple) et d'utiliser le module `Modernizr` de `npm` :

<cli>root@muffin:~# cd ~/modernizr root@muffin:~/modernizr# modernizr -c modernizr-config.json Modernizr build saved to ~/modernizr/modernizr.js</cli>

Ce qui a pour effet de compiler et de télécharger le fichier `modernizr.js` correspondant aux données contenues dans le fichier `.json`. Par la suite, il suffit de modifier ce dernier et de relancer la commande `modernizr` ci-dessus pour télécharger une nouvelle version.