02

Introdução

Introdução

Em algum momento em sua carreira de desenvolvedor você já deve ter executado uma série de tarefas mesmo com o projeto pronto como empacotá-lo para depois enviá-lo via ftp.

Por exemplo, se você é um desenvolvedor front-end já deve ter se preocupado em *minificar* e *concatenar* seus scripts, inclusive já deve ter experimentado o uso de algum pré-processador como LESS (http://lesscss.org/).

A qualidade e quantidade de passos variam de acordo com as necessidades do seu projeto e essas necessidades acabam gerando um **fluxo (workflow)** identificável e que muitas vezes é documentado para ser usado por toda a equipe.

O problema é que tudo que é feito pelo ser humano está sujeito a erro. Por mais que tenhamos um manual nos dizendo o que fazer, nada nos impede de pularmos um dos passos, o que pode afetar diretamente o resultado final.

Grunt: automação de build front-end

Para solucionar problemas como esse, foram criadas no mercado ferramentas de construção (build) de projetos como Ant (http://ant.apache.org/), Gradle (http://www.gradle.org/) e Maven (http://maven.apache.org/), mas há uma que nasceu voltada especialmente para programadores front-end: o **Grunt.**

O Grunt (http://gruntjs.com/) é um *task runner* totalmente feito em JavaScript tornando-o atrativo no mundo front-end, inclusive ele possui um acervo com mais de 2.000 plugins suportando as mais diversas funcionalidades.

Node.js e instalação

O Grunt é escrito em JavaScript mas não executa no browser. Ele executa no terminal usando o **Node.js**.

O Node.js (http://nodejs.org/) é uma plataforma construída sob a máquina virtual V8, aquela utilizada no Google Chrome. Ela permite o uso da linguagem JavaScript fora do browser.

A instalação do Node.js é feita baixando seu instalador diretamente em sua página (http://nodejs.org/). Logo de cara você verá um grande botão com o texto "install".

Não se preocupe, existe uma versão para cada sistema operacional e clicando no botão será baixada a correta. Depois, basta executar o instalador e seguir as instruções até o fim.

O projeto

Ao longo do treinamento trabalharemos com este projeto (https://s3.amazonaws.com/caelum-online-public/grunt/01-projeto.zip). É uma página simples que possui um diretório chamado 'public' que contém uma página, algumas imagens, arquivos CSS e JavaScript. Você já pode baixar o projeto descompactando-o logo em seguida.

Instalando o Grunt através do NPM e do package.json

O Grunt nada mais é do que um módulo do Node.js. Para instalarmos um módulo do Node.js, usamos o npm (https://www.npmjs.org/), o gerenciador de pacotes do Node.js.

Por uma questão de organização, o Node.js pode tomar nota de todos os módulos de que sua aplicação depende num arquivo chamado **package.json**. Podemos criá-lo através do próprio npm executando o seguinte comando **dentro da pasta projeto**:

```
npm init
```

Um assistente fará perguntas como o nome do projeto, sua versão entre outras. **Você pode dar ENTER para todas elas que um padrão será adotado**. Por exemplo, o nome do projeto será o nome da pasta na qual o comando foi executado. O resultado final será o arquivo *package.json*.

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (projeto)
version: (0.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to package.json:

{
  "name": "projeto",
  "version": "0.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes)
```

Agora que temos o arquivo package.json podemos instalar o Grunt através da linha de comando:

```
npm install grunt --save-dev
```

O comando anterior baixará a versão mais atual do Grunt gravando-a dentro da pasta **node_modules**. Esta pasta guarda todos os módulos usado pelo seu projeto. O parâmetro `--save-dev` adiciona a dependência no arquivo *package.json*:

```
// package.json
{
  "name": "projeto",
  "version": "0.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "grunt": "^0.4.5"
  }
}
```

Para o Grunt funcionar, ainda precisamos instalar seu cliente em linha de comando, o **grunt-cli**:

```
npm install grunt-cli -g
```

É importante que você tenha permissão de administrador já que estamos instalando o módulo globalmente através do parâmetro `-g`. Este módulo nos permite chamar o comando 'grunt' em qualquer local em nosso terminal.

Repare que não usamos o parâmetro `--save-dev`. Não nos importa qual versão do cliente de linha de comando estamos usando, apenas a versão do Grunt do projeto.

Instalar o Grunt por projeto permite sua atualização sem afetar os demais projetos, algo que não seria possível se o Grunt fosse instalado globalmente.

Outra vantagem de salvarmos nossas dependências em *package.json* é que você pode compartilhar seu projeto sem a pasta *node_modules*, que deve ser baixada novamente através do comando:

```
npm install
```

O comando lerá de *package.json* todos os módulos de que sua aplicação depende baixando-os todos de uma só vez, levando em consideração cada versão utilizada.

O arquivo Gruntfile.js

O arquivo **Gruntfile.js** é onde configuramos tarefas (tasks). Todo o código ficará envolto por uma função wrapper que recebe como parâmetro o objeto do Grunt.

```
/* recebe o objeto grunt como parâmetro*/
module.exports = function(grunt) {
   grunt.initConfig({
        /*  suas tasks aqui */
  });
}
```

Mas quem chamará essa função passando o objeto? Nós mesmos, quando executarmos o comando `grunt` no terminal. Quando executado na mesma pasta do arquivo Gruntfile.js, ele se encarregará de chamar a função wrapper definida no arquivo passando uma referência para o objeto grunt.

Antes de pensarmos em qualquer tarefa, precisamos garantir que todas elas sejam aplicadas numa cópia da pasta 'public' de nosso projeto, garantindo assim a integridade dos arquivos originais.

Poderíamos até tentar fazer isso usando os comandos padrões do Grunt mas, ainda assim, precisaríamos varrer recursivamente a pasta que desejamos copiar e executar uma série de comandos fragmentados.

Para esta tarefa, tão corriqueira, podemos usar um plugin do Grunt que também é um módulo do Node.js.

Instalando nosso primeiro plugin

O primeiro plugin do Grunt que utilizaremos será o grunt-contrib-copy (https://github.com/gruntjs/grunt-contrib-copy). Para instalá-lo via npm:

```
npm install grunt-contrib-copy --save-dev
```

Pronto, mas isso ainda não é suficiente. Apesar do npm ter criado a pasta grunt-contrib-copy dentro de node_modules, o plugin ainda não é enxergado pelo Grunt. Para que ele seja enxergado precisamos registrá-lo no Gruntfile.js através da função **grunt.loadNpmTasks**:

```
/* recebe o objeto  grunt como parâmetro*/
module.exports = function(grunt) {
   grunt.initConfig({
        /*  suas tasks aqui */
  });

  grunt.loadNpmTasks('grunt-contrib-copy');
};
```

Repare que a instrução anterior não faz parte da configuração de tarefas, logo, fica fora da função `grunt.initConfig` . Por falar em tarefas, que tal configurarmos a tarefa *copy*?

Configurando tarefas (tasks) do Grunt

Plugins no Grunt são configurados através da função **grunt.initConfig({ })**:

```
   grunt.initConfig({
        plugin1: {
           /* configurações do plugin1 */
         },
        plugin2: {
          /* configurações do plugin2 */
        }
  });
```

Repare que a função recebe um **objeto JavaScript** . Este objeto tem como propriedades os nomes das `tasks` que desejamos configurar e como valor suas configurações. Sabemos pela documentação do `grunt-contrib-copy` que

propriedade que representa a task deste plugin é **copy**. Assim temos:

```
 grunt.initConfig({
      copy: {/* configurações de grunt-contrib-copy*/}
});
```

Targets

Cada tarefa do Grunt pode ter alvos (targets) diferentes, que podem ser entendidos como subtarefas. Por exemplo, podemos querer copiar a pasta public inteiramente ou apenas a pasta public/css

```
 grunt.initConfig({
      copy: {
          tudo: {/* copia a pasta public */},
          css: {/* copia a pasta public/css apenas */}
      }
});
```

O exemplo anterior não tem as tasks totalmente configuradas, mas se tivessem prontas, poderíamos executar no terminal:

```
grunt copy
```

O comando anterior executará a task `copy` e seus targets `tudo` e `css`. Mas se quisermos rodar apenas um target específico? Podemos qualificar a task indicando qual target queremos executar:

```
grunt copy:tudo
```

Agora que entendemos o mecanismo de task e target, vamos configurar nossa task copy:

```
module.exports = function(grunt) {
   grunt.initConfig({
        copy: {
              public: {
                   cwd: 'public',
                   src: '**',
                   dest: 'dist',
                   expand: true
              }
         }
   });

   grunt.loadNpmTasks('grunt-contrib-copy');
};
```

Vamos entender os parâmetros da task copy:

- **expand**: quando `true` ativa o **mapeamento dinâmico**. No lugar de definirmos o nome de cada arquivo e seu destino, indicamos o diretório de trabalho (cwd) a origem (src) e o destino (desc). Caso contrário precisaríamos fazer da seguinte forma:

```
copy: {
   public: {
       files: [
           {src: 'public/index.html', dest: 'dist/index.html'},
           {src: 'public/css/index.css', dest: 'dist/css/index.css'},
           /* demais arquivos */
       ],
   }
}
```

- **cwd**: diretório padrão (current work directory) no qual as demais propriedades se basearão.
- **src**: arquivos que desejamos copiar. Usamos o *globbing pattern* `**` para copiar todos os arquivos e diretórios.
- **dest**: pasta de destino. Em nosso caso, a pasta `dist`. É criada caso não exista.

Agora que temos tudo configurado, podemos rodar a task copy através do grunt:

```
$ grunt copy
Running "copy:public" (copy) task
Created 4 directories, copied 5 files

Done, without errors.
```

A pasta `dist` foi criada, sendo uma réplica da pasta public.

Registrando tarefa atalho

Tudo funciona, mas imagine o seguinte cenário: alguém apagou algum arquivo do projeto. Nossa cópia precisa refletir o projeto, logo, rodamos novamente o comando `grunt copy`. O problema é que ele conterá o arquivo que foi apagado do projeto original e isso é um problema.

Precisamos apagar a pasta 'dist' antes de rodarmos novamente nossa task 'copy'. Para isso existe a task **clean**. Instalamos o plugin através do comando:

```
npm install grunt-contrib-clean --save-dev
```

Não podemos nos esquecer de registrá-lo em nosso arquivo :

```
grunt.loadNpmTasks('grunt-contrib-clean');
```

E por fim configurar a task:

```
module.exports = function(grunt) {
   grunt.initConfig({
        copy: {
             public: {
                  cwd: 'public',
                  src: '**',
                  dest: 'dist',
                  expand: true
             }
         },
```

```
        clean: {
             dist: {
                 src: 'dist'
             }
        }
  });

  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');
};
```

Usamos o nome `clean` porque é este o nome da task fornecido por sua documentação (https://github.com/gruntjs/grunt-contrib-clean). Escolhemos o nome `dist` para seu target, que recebe como parâmetro um objeto com a propriedade `src`. Nele definimos o diretório que será apagado.

Testando a task clean:

```
$ grunt clean
Running "clean:dist" (clean) task
Cleaning dist...OK

Done, without errors.
```

Agora basta rodar as tasks na ordem e torcer para não esquecer:

```
grunt clean
```

```
grunt copy
```

Você pode executar as duas de uma vez:

```
grunt clean copy
```

Repare que estamos querendo automatizar nossas tarefas e ter que lembrar de executar em ordem as duas não se coaduna com nosso objetivo.

Para resolver problemas como esse, o Grunt permite registrar novas tasks que chamam outras na sequência que definirmos. Fazemos isso através da função **grunt.registerTask**.

```
grunt.registerTask('dist', ['clean', 'copy']);
```

Repare que a função acima recebe como primeiro parâmetro o nome da nossa task. O segundo é um array com o nome de todas as tasks já configuradas pelo Grunt. A ordem é importante, pois a primeira será executada antes da segunda e por ai vai.

Nosso script final fica:

```
module.exports = function(grunt) {
   grunt.initConfig({
        copy: {
             public: {
```

```
                    cwd: 'public',
                    src: '**',
                    dest: 'dist',
                    expand: true
                }
            },
            clean: {
                dist: {
                    src: 'dist'
                }
            }
    });

    grunt.registerTask('dist', ['clean', 'copy']);

    grunt.loadNpmTasks('grunt-contrib-copy');
    grunt.loadNpmTasks('grunt-contrib-clean');
};
```

Agora basta executarmos no terminal:

```
$ grunt dist
Running "clean:dist" (clean) task

Running "copy:public" (copy) task
Created 4 directories, copied 5 files

Done, without errors.
```

O Grunt ainda permite registrarmos a task 'default'. Ela será executada apenas através do comando 'grunt'. Modificando nosso script:

```
module.exports = function(grunt) {
    grunt.initConfig({
        copy: {
            public: {
                cwd: 'public',
                src: '**',
                dest: 'dist',
                expand: true
            }
        },
        clean: {
            dist: {
                src: 'dist'
            }
        }
    });

    grunt.registerTask('dist', ['clean', 'copy']);
    grunt.registerTask('default', ['dist']);

    grunt.loadNpmTasks('grunt-contrib-copy');
```

```
  grunt.loadNpmTasks('grunt-contrib-clean');
};
```

Agora basta executar no terminal:

```
$ grunt
Running "clean:dist" (clean) task
Cleaning dist...OK

Running "copy:public" (copy) task
Created 4 directories, copied 5 files

Done, without errors.
```

Agora já temos nossa configuração mínima para podemos realizar coisas incríveis com o Grunt.