MAD

Small and Fast Build Tool for Erlang Apps

Contents

1 MAD: Erlang Build and Deploy Tool

1.1 History

We came to conclusion that no matter how perfect your libraries are, the comfort and ease come mostly from developing tools. Everything got started when Vladimir Kirillov[1] decided to replace Rusty's sync beam reloader. As you know sync uses filesystem polling which is neither energy-efficient nor elegant. Also sync is only able to recompile separate modules while common use-case in N2O is to recompile DTL templates and LESS/SCSS stylesheets. That is why we need to recompile the whole project. That's the story how active[2] emerged. Under the hood active is a client subscriber of fs[3] library, native filesystem listener for Linux, Windows and Mac.

De-facto standard in Erlang world is rebar. We love rebar interface despite its implementation. First we plugged rebar into active and then decided to drop its support, it was slow, especially in cold recompilation. It was designed to be a stand-alone tool, so it has some glitches while using as embedded library. Later we switched to Makefile-based build tool otp.mk[4].

The idea to build rebar replacement was up in the air for a long time. The best minimal approach was picked up by Sina Samavati[5], who implemented the first prototype called 'mad'. Initially mad was able to compile DTL templates, YECC files, escript (like bundled in gproc), also it had support for caching with side-effects. In a month I forked mad and took over the development under the same name.

Listing 1: Example of building N2O sample

```
                                Cold      Hot
rebar get-deps compile          53.156s   4.714s
mad deps compile                54.097s   0.899s
```

[1] https://github.com/proger
[2] https://github.com/synrc/active
[3] https://github.com/synrc/fs
[4] https://github.com/synrc/otp.mk
[5] https://github.com/s1n4

Listing 2: Example of building Cowboy

```
                                Hot
make (erlang.mk)                2.588s
mad compile                     2.521s
```

1.2 Introduction

We were trying to make something minimalistic that fits out Web Stack[6]. Besides we wanted to use our knowledge of other build tools like lein, sbt etc. Also for sure we tried sinan, ebt, Makefile-based scripts.

Synrc mad has a simple interface as follows:

```
BNF:
    invoke := mad params
    params := [] | run params
       run := command [ options ]
   command := app | lib | deps | compile | bundle
              start | stop | repl
```

It seems to us more natural, you can specify random commands set with different specifiers (options).

1.3 Several Types of Packaging

The key feature of mad is ability to create single-file bundled web sites. Thus making dream to boot simpler than node.js come true. This target escript is ready to run on Windows, Linux and Mac.

1.4 Deployment Options

mad is also supposed to be a deploy tool with ability to deploy not only to our resources like Erlang on Xen, Voxoz (LXC/Xen) but also to Heroku and others.

[6] https://github.com/synrc

1.5 OTP Compliant

mad supports rebar umbrella project structure. Specifically two kinds of directory layouts:

1.6 Fast Apps Ordering

As you may know you can create OTP releases with reltool (rebar generate) or systools (relx). mad currently creates releases with relx but is going to do it independently soon. Now it can only order applications.

1.7 Tiny Size

And the good part:

	Sources	Binary
mad	567 LOC	39 KB
rebar	7717 LOC	181 KB

2 Setup

2.1 Installing Binary

Fresh version of mad included as a binary in its primary github repository:

```
# curl -fsSL https://raw.github.com/synrc/mad/master/mad > mad \
          && chmod +x mad \
          && sudo cp /usr/local/bin
```

Or you may want to add mad to your PATH.

2.2 Compiling Sources

If you want you can compile mad by yourself:

```
# git clone http://github.com/synrc/mad \
         && cd mad \
         && make

# cat Makefile
  default:
      ./mad cle dep com bun mad
```

Note that mad uses mad to build mad. It's mad.

2.3 Creating a sample N2O project

mad also comes with N2O templates. So you can bootstrap a N2O-based site just having a single copy of mad binary.

```
# mad app sample
# cd sample
# mad deps compile plan bundle web_app
```

After that you can just run escript web_app under Windows, Linux and Mac and open http://localhost:8000[7].

```
C:\> escript web_app
Applications: [kernel,stdlib,crypto,cowlib,ranch,
               cowboy,compiler,syntax_tools,
               erlydtl,gproc,xmerl,n2o,sample,
               fs,active,mad,sh]
Configuration: [{n2o,[{port,8000},
                      {route,routes}]},
                 {kvs,[{dba,store_mnesia},
                      {schema,[kvs_user,
                               kvs_acl,
                               kvs_feed,
                               kvs_subscription]}]}]
Erlang/OTP 17 [erts-6.0] [64-bit] [smp:4:4]
              [async-threads:10] [kernel-poll:false]

Eshell V6.0  (abort with ^G)
1>
```

[7] http://localhost:8000

3 Configuration File

3.1 rebar.config

mad uses **rebar.config** filename to load mad configuration. Despite mad is no fully rebar compatible (e.g. it can't uses rebar plugins, ports compilation is rather different, etc), it uses its name to achive certail level of compatibility.

3.2 deps

deps is the core option of mad. It says which OTP applications shold be used and where they could be found. Yoy may also specify versions. Here is simpliest example:

Listing 3: deps Option

```
{deps, [
        {kvs,   ".*", {git,"git://github.com/synrc/kvs"}},
        {forms, ".*", {git,"git://github.com/spawnproc/forms"}}
]}.
```

3.3 deps_dir

To specify where deps should be stored after fetching inside your application you use deps_dir option:

Listing 4: deps_dir Option

```
{deps_dir, "deps"}.
```

3.4 sub_dirs

If your application consist of more than one src directory, you may specify all of the sub-applications. Each sub-application should be valid OTP application with its own rebar.config configuration file.

```
{sub_dirs,["apps"]}.
```

3.5 lib_dirs

To use include directive across your sub-applications you should specify the **lib_dirs** directories which will be settled as include directories during compilation.

```
{lib_dirs,["apps"]}.
```

E.g. you have my_app and my_server applications inside apps directory and you including HRL file from my_server application from ap_app application:

```
-module(my_app).
-include_lib("my_server/include/my_server.hrl").
```

4 Commands

Synrc mad has a simple interface as follows:

```
BNF:
    invoke := mad params
    params := [] | run params
       run := command [ options ]
   command := app | lib | deps | compile | bundle
              start | stop | repl
```

It seems to us more natural, you can specify random commands set with different specifiers (options).

4.1 deps

4.2 compile

4.3 plan

4.4 repl

4.5 bundle

4.6 app

5 Dependencies

5.1 OTP Compliant

mad supports rebar umbrella project structure. Specifically two kinds of directory layouts:

Listing 5: Solution

```
+-- apps
+-- deps
+-- rebar.config
+-- sys.config
```

Listing 6: OTP Application

```
+-- deps
+-- ebin
+-- include
+-- priv
+-- src
+-- rebar.config
```

5.2 Apps Ordering

As you may know you can create OTP releases with reltool (rebar generate) or systools (relx). mad currently creates releases with relx but is going to do it independently soon. Now it can only order applications.

```
# mad plan
Ordered: [kernel,stdlib,mnesia,kvs,crypto,cowlib,ranch,
          cowboy,compiler,syntax_tools,erlydtl,gproc,
          xmerl,n2o,n2o_sample,fs,active,mad,rest,sh]
```

6 Bundles

6.1 Apps Ordering

Before starting an application which could be packed within three forms: Single-File Bundle, Release or OTP.MK Folder.

```
# mad plan
Ordered: [kernel,stdlib,mnesia,kvs,crypto,cowlib,ranch,
          cowboy,compiler,syntax_tools,erlydtl,gproc,
          xmerl,n2o,n2o_sample,fs,active,mad,rest,sh]
```

6.2 Single-File Bundles with MAD

The key feature of mad is ability to create single-file bundled web sites. Thus making dream to boot simpler than node.js come true. This target escript is ready to run on Windows, Linux and Mac.

To make this possible we implemented a zip filesytem inside escript. mad packages priv directories along with ebin and configs. You can redefine each file in zip fs inside target escript by creation the copy with same path locally near escript. After launch all files are copied to ETS. N2O also comes with custom cowboy static handler that is able to read static files from this cached ETS filesystem. Also bundle are compatible with active online realoading and recompilation.

E.g. you main create a single file site with:

```
# mad bundle app_name
```

app_name shoul be the same as a valid Erlang module, with app_-module:main/1 function defined, which will boot up the bundle. This function could be like that:

```
-module(app_name).

main(Params) ->
     RebarConfig = [],
```

```
mad_repl:main(Params,RebarConfig).
```

6.3 Releases with RELX

As you may know you can create OTP releases with reltool (rebar generate) or systools (relx). mad currently creates releases with relx but is going to do it independently soon. Now it can only order applications.

```
# mad release
```

6.4 Folders with OTP.MK

OTP.MK is a tiny 50 lines Makefile that allows to start your set of application using run_erl and to_erl tools from OTP distribution. We use that way in poduction. This is the best option also in development mode because all directory structure is open and mutable, so you can reload modified files and perform recompilation on the fly.

It uses the original code to fast resolve dependencies into the right boot sequence to start.

```
# make console
```