Vendoring with Quicklisp, Make, Git, and Nix

Max Rottenkolber <max@mr.gy>

Sunday, 19 August 2018

I have a pet project called *Athens* (https://athens.mr.gy) (*source* (https://github.com/eugeneia/athens)), it is a server application built with *CCL* (https://ccl.clozure.com/). What it does is entirely irrelevant for the topic at hand. What follows are my thoughts on dependency management for Lisp applications, and my experience with some tools—old and new—that solved my woes.

Dependency hell

For being a seemingly simple application, Athens has an awful lot of dependencies of different kinds. A dozen primary dependencies include both third-party systems from *Quicklisp* (https://www.quicklisp.org/beta/), as well as homegrown systems not in Quicklisp. These primary dependencies pull in countless secondary dependencies from Quicklisp, which in turn depend on C libraries provided by the OS such as OpenSSL and LibXML2. Ouch.

I want building Athens to be trivial and predictable. No chasing down dependencies. I also want to be in control of when dependencies are upgraded, and be able to reproduce build and test environments faithfully. I am fine with only supporting the Unix-like operating systems supported by CCL, so platform support is restricted to those.

Making things easy

Athens has a human-readable *Makefile* (https://github.com/eugeneia/athens/blob/mr.gy/blog/lisp-vendoring-quicklisp-nix/Makefile) that describes how the application is built. You can `cd` into its source directory and type `make` to produce an executable (which includes the CCL kernel and the Athens Lisp image) which will act like a reasonable Unix citizen.

```
$ cd ~/git/athens/
$ make
mkdir bin
ccl -Q -b -n -l quicklisp/setup.lisp -l build/athens.lisp
To load "athens":
  Load 1 ASDF system:
    athens
; Loading "athens"
...
du -h bin/athens
65M bin/athens
$ bin/athens --help
Usage: athens init <configuration>
       athens start <configuration>
       athens -h|-help|--help
```

Building Athens.

Right now, its build dependencies (minus ubiquitous core utilities and GNU Make) are CCL, OpenSSL and LibXML2, the latter two of which are also runtime dependencies that are dynamically loaded as shared objects.

```
 bin/athens: quicklisp bin build/athens.lisp $(SOURCE_OBJECTS)
ccl -Q -b -n -l quicklisp/setup.lisp -l build/athens.lisp
du -h bin/athens
```

Rule to build Athens.

The Make rule to build Athens invokes `ccl` to run the actual *build script* (https://github.com/eugeneia/athens/blob/mr.gy/blog/lisp-vendoring-quicklisp-nix/build/athens.lisp) which is written in Lisp. It also loads a project-local Quicklisp installation! Athens ships with Quicklisp, and includes all its Lisp dependencies checked into the repository as part of the Quicklisp installation. The release manager (me) makes sure all is in order on the `master` branch.

```
 ASD = $(shell find src/ -regex '[^\#]*\.asd' -printf '%p ')
 VENDOR_ASD = $(shell find lib/ -regex '[^\#]*\.asd' -printf '%p ')
 quicklisp:
ccl -Q -b -n -l lib/quicklisp/quicklisp.lisp \
-e '(quicklisp-quickstart:install :path "quicklisp/")' \
-e '(quit)'
for asd in $(ASD) $(VENDOR_ASD); \
do ln -s -v ../../$$asd quicklisp/local-projects/; done
```

Rule to vendor in a Quicklisp dist. Note: there is another, officially supported way to do essentially the same thing via Quicklisp's *bundle-systems* (http://blog.quicklisp.org/2015/04/new-feature-quicklisp-library-bundles.html).

Athens vendors a bunch of homegrown and pinned dependencies (currently `xmls-1.7`, because its API changed and I have not found the time to restore compatibility in my downstream library, yet) in `lib/`. The remaining third-party dependencies are managed via the fabulous Quicklisp. The above rule installs Quicklisp into the source tree and links all non-Quicklisp systems to its `local-projects/`. In order to upgrade to a new Quicklisp dist I would do

```
$ make tidy && make
```

…and check in the resulting `quicklisp/` tree into the repository (after testing that the resulting Athens binary built with the new dist works as expected.) To avoid excessive repository bloat, Athens has some Quicklisp specific patterns in its `.gitignore` file.

```
quicklisp/tmp
quicklisp/dists/quicklisp/archives
```

Patterns in `.gitignore` specific to Quicklisp.

I manage the vendored dependencies in `lib/` (including *quicklisp-bootstrap* (https://github.com/quicklisp/quicklisp-bootstrap)) with *git-subtree* (https://raw.githubusercontent.com/git/git/v2.18.0/contrib/subtree/git-subtree.txt) when possible.

Nixing the build

In addition to its `Makefile`, Athens comes with a *default.nix* (https://github.com/eugeneia/athens/blob/mr.gy/

blog/lisp-vendoring-quicklisp-nix/default.nix) file that enables it to be built by *Nix* (https://nixos.org/nix/). To build Athens using Nix you can do

```
$ nix-build path/to/athens
```

Build outputs will be in `./result`.

…which performs the build in tidy environment with all build dependencies provided, and produces a so-called Nix derivation of Athens. Likewise, Athens (and its runtime dependencies) can be installed in a Nix environment via

```
$ nix-env -f path/to/athens -i
```

We can also interactively test Athens and its build in a controlled environment:

```
$ cd path/to/athens
$ nix-shell
$ runHook preBuild
$ make
```

By default, Nix will use the dependencies from the active system or user environment, but it does not restrict you to your current system. On of its cool features is that you can reproduce the build environment for any previous or future version of *nixpkgs* (https://github.com/NixOS/nixpkgs) (a Git repository of Nix expressions for every piece of software distributed by Nix.) To test Athens with the most recent nixpkgs you could do

```
$ nix-shell -I nixpkgs=https://github.com/NixOS/nixpkgs/archive/master.tar.gz
```

Note: the argument to `-I` could also be a local fork of *nixpkgs*, so its easy to vendor and customize dependency configurations.

Athens' `default.nix` contains a few Lisp specific bits that warrant an explanation. It starts out by declaring its build dependencies. Since Athens is a CCL application this includes `ccl` and its dependencies `openssl` and `libxml2`, as well as a utility `makeWrapper` provided by Nix (to be explained later.)

```
buildInputs = [ makeWrapper ccl openssl libxml2 ];
```

Runtime dependencies are declared by inheriting `openssl` and `libxml2` (along with the version string for this build of Athens) into the derivation.

```
inherit version openssl libxml2;
```

Note that CCL is not a runtime dependency of Athens since the Lisp kernel will be included with the executable produced during build.

Our `preBuild` hook specifies some steps to ensure a clean build. CCL will try to load shared objects for OpenSSL and LibXML2 provided by the operating system (on *NixOS* (https://nixos.org/) these won't be present in the usual locations at all.) To ensure that it will only use the shared objects specified for the build we set `LD_LIBRARY_PATH` accordingly. The remaining bits tell ASDF where to cache compiled Lisp code during the build by setting `XDG_CACHE_HOME` to a suitable location, and to cease its attempt to load system and user configurations for the source registry (which could mess with the build) via `CL_SOURCE_REGISTRY`.

```
preBuild = ''
  make clean
  export LD_LIBRARY_PATH=${lib.makeLibraryPath [ openssl libxml2 ]}
  export XDG_CACHE_HOME="$TMP/.cache"
  export CL_SOURCE_REGISTRY="(:source-registry :ignore-inherited-configuration)"
'';
```

Nix will implicitly build software using `make`, and that works just fine for Athens so we can get away without declaring a `buildPhase`. Athens' `Makefile` does not have an `install` target though, a custom `installPhase` is needed. This is also where `makeWrapper` comes into play. When the CCL kernel loads the Athens Lisp image, it will dynamically open the shared libraries used by Athens (take a look at CCL's *REVIVE-SHARED-LIBRARIES* (https://github.com/Clozure/ccl/blob/1.11/level-0/l0-cfm-support.lisp#L535-L552) function for the gritty details.) To make sure it finds the share objects specified in the derivation, the Athens executable is replaced by a shell script that again sets `LD_LIBRARY_PATH` and then `exec`s the actual executable.

```
installPhase = ''
  mkdir -p $out/bin
  cp bin/athens $out/bin/athens.orig
  makeWrapper $out/bin/athens.orig $out/bin/athens \
      --suffix LD_LIBRARY_PATH : $LD_LIBRARY_PATH
'';
```

Finally, we need to inhibit Nix's default behavior of stripping ELF symbols from the resulting executables. This just happens to lobotomize the executable produced by CCL.

```
dontStrip = true;
```

So where does this leave us? If you are a user of Nix then you can let Nix figure out the dependencies and produce a clean build of Athens easily. Or put alternatively, Athens is left with only a single dependency: Nix. Of course, it is still possible to build and run Athens without Nix by manually resolving its its dependencies, and using the Makefile directly.