Packaging with Homebrew

Ben Morgan

THE UNIVERSITY OF WARWICK

What Does Homebrew Do?

Homebrew installs the stuff you need that Apple didn't.

```
$ brew install wget
```

Homebrew installs packages to their own directory and then symlinks their files into `/usr/local`.

```
$ cd /usr/local
$ find Cellar
Cellar/wget/1.16.1
Cellar/wget/1.16.1/bin/wget
Cellar/wget/1.16.1/share/man/man1/wget.1

$ ls -l bin
bin/wget -> ../Cellar/wget/1.16.1/bin/wget
```

Homebrew won't install files outside its prefix, and you can place a Homebrew installation wherever you like.

Trivially create your own Homebrew packages.

```
$ brew create https://foo.com/bar-1.0.tgz
Created /usr/local/Library/Formula/bar.rb
```

Ruby-based package manager http://brew.sh

```
… Get Homebrew …
$ git clone https://github.com/Homebrew/homebrew brew.git
… Add brew.git/bin to your PATH …
… Add additional repository (“tap” in brewspeak) …
$ brew tap homebrew/science

… install ROOT with fftw …
$ brew install root –with-fftw
$ root

… install Geant4 with GDML …
$ brew install geant4 –with-gdml

… some time later …
$ brew update
$ brew upgrade
```

User/Installer Case

Easy to use workflow, identical to that of other systems (MacPorts, yum, apt)

$ brew create http://proj-clhep.web.cern.ch/…/clhep-2.2.0.4.tgz
… opens ‘clhep.rb’, edit it…

```
class Clhep < Formula
  homepage "http://proj-clhep.web.cern.ch/proj-clhep/"
  url "http://proj-clhep.web.cern.ch/proj-clhep/DISTRIBUTION/tarFiles/clhep-2.2.0.4.tgz"
  sha1 "60a291b940fdc78bea4aaeaffc147cc25a42cfef"

  bottle do
    cellar :any
    sha1 "5466fbee57b366a41bbcec814614ee236e39bed8" => :yosemite
    sha1 "bde270764522e4a1d99767ca759574a99485e5ac" => :mavericks
    sha1 "e77d0e5f516cb41ac061e1050c8f37d0fb65b796" => :mountain_lion
  end

  depends_on "cmake" => :build
  option :cxx11

  def install
    ENV.cxx11 if build.cxx11?
    mkdir "clhep-build" do
      args = std_cmake_args
      args << "-DCLHEP_BUILD_CXXSTD=c++11" if build.cxx11?
      system "cmake", "../CLHEP", *args
      system "make", "install"
    end
  end
end
```

Can use Binary Packages known as “bottles”

Note support for C++ Standard as dependency

Packager Case I

“Build Protocol” is a simple Ruby script called a Formula

```
… Test install package dropping to interactive shell on
… error
$ brew install -vd clhep

… Can also do full interactive install with local git
… repo for patches …
$ brew install -interactive -git clhep

… when everything's working and the Formula is ready …
$ git add Library/Formula/clhep.rb
$ git commit -m "clhep: new formula"

… A new version arrives …
$ brew edit clhep
…
$ git commit -m "clhep: new version A.B.C.D"
```

Packager Case 2

Interactive testing, Git control of package histories

Evaluation for SuperNEMO and Dune

- SuperNEMO approach: Fork homebrew, adapt formula to requirements, rolling release with git tags to snapshot for production points (very early days here)
  - https://github.com/SuperNEMO-DBD/cadfaelbrew
- DUNE approach: Provide tap containing custom formulae(e.g. Art), otherwise use upstream (e.g. gcc)
  - https://github.com/drbenmorgan/homebrew-dunebrew
- So far so good, but still lots to look at and try out

Why (Not) Homebrew?

- Works out the box on Mac and Linux
- *Extremely* easy to use and add new packages
- Good support for build variants and C++ Standards
- Only provides a single rolling release
- Doesn’t directly support git tags or rollback on versions
- Binary packages not completely relocatable(*)

On Build Protocols

- == File(s) specifying package's
  - Metadata (name, version, dependencies etc)
  - Steps required to get, patch, configure, build, test install
- Examples
  - RPM Specfile
  - Homebrew Formula

```
“hello.spec”

Name: hello
Version: 2.10
Source0: <base>/%{name}-%{version}.tar.gz
BuildRequires: gettext

%prep

%build
%configure
make

%install
%make_install
…
```

```
“hello.rb”

require “formula”
class Hello < Formula
  url  <base>/hello-2.10.tar.gz
  depends_on “gettext” => :build

  def install
    system “./configure”, “-prefix=#{prefix}”
    system “make”, “install”
  end
```

HSF Protocol?

- “build.sh” on HSF GitHub: An “adaptor” layer between package manager and build tool?

- However, that’s **exactly** what Specfiles/Formulas are.

- Likely to end up writing an adaptor for an adaptor because assumptions of a “build.sh” won't match up with all packaging systems

With “build.sh”

```
require “formula”
class MyPkg < Formula
  url  <base>/mypkg-1.0.0.tar.gz

  depends_on “zlib” => :build

  def install
    ENV[“HEP_COMPILER”] = #{ENV.cxx}
    ENV[“HEP_SOURCEDIR”] = #{buildpath}
    ENV[“HEP_BUILDROOT”] = #{buildpath}
    ENV[“HEP_INSTALLROOT”] = #{prefix}
    ENV[“HEP_ARCH”] = hardware.is_64_bit? ? “x86_64”
    ENV[“ZLIB_ROOT”] = #{opt_prefix}/zlib
    … and so on …

    system “./build.sh”
 end
```

Without…

```
  def install
    # This is essentially what “build.sh” does…
    system “./configure”, “–prefix=#{prefix}”,
           “–zlib-root=#{opt_prefix}/zlib”
    system “make”, “install
  end
```