Python Packaging and Distribution: bdist_rpm

Matt Behrens
Grand Rapids Python User Group
March 17, 2014

A lightning introduction:

How an RPM is made

From code to RPM

- Source code tarballs and patches
- .spec file
- Package metadata: version, dependencies, etc.
- Scriptlets: scripts that run at various times during a package's lifecycle

A basic .spec

```
Version:        2.8
Release:        1%{?dist}
Summary:        The "Hello World" program from GNU
License:        GPLv3+
URL:            http://ftp.gnu.org/gnu/%{name}
Source0:        http://ftp.gnu.org/gnu/%{name}/%{name}-%{version}.tar.gz
BuildRequires:  gettext
Requires:       info

%description
The "Hello World" program, done with all bells and whistles of a proper FOSS
project, including configuration, build, internationalization, help files, etc.

%prep
%setup -q

%build
%configure
make %{?_smp_mflags}

%install
%make_install
%find_lang %{name}
rm -f %{buildroot}/%{_infodir}/dir
```

A basic .spec (cont'd)

```
%post
/sbin/install-info %{_infodir}/%{name}.info %{_infodir}/dir || :

%preun
if [ $1 = 0 ] ; then
/sbin/install-info --delete %{_infodir}/%{name}.info %{_infodir}/dir || :
fi

%files -f %{name}.lang
%doc AUTHORS ChangeLog COPYING NEWS README THANKS TODO
%{_mandir}/man1/hello.1.gz
%{_infodir}/%{name}.info.gz
```

From code to RPM, cont'd.

- rpmbuild
  - Sources, .spec file → source RPM (SRPM)
  - Can also build binary RPM… though I don't recommend it
- mock (+ setarch for x86_64 → i386 builds)
  - Source RPM → binary RPM
  - Builds in chroot environment to insure all dependencies are adequately specified

Binary RPM

- These are what you've probably dealt with
- i386, x86_64: contains binaries for a specific architecture
- noarch: can run on any architecture

Now forget (most) of that

bdist_rpm

- Go from setup.py to binary RPM in one step
- You will at least need python-devel and rpm-build on RHEL
- Works on most simple setup.py files out-of-the-box
- Though some workarounds may be needed…

optimize=1

- Needed by older versions of RHEL 5
- Instructs distutils to build the .pyo files that rpm expects
- You may see this in several older packages; it's an old bug
- Put it in setup.cfg like so:

```
[install]
optimize = 1
```

Installing non-modules

- Scripts
  - Supply the local paths of the scripts in a list
    - Will be installed to /usr/bin
- Data files
  - Supply a list of tuples of install directory, list of local paths

Installing non-modules: example

```
setup(name="hello",
      version=version,
      packages=["hello"],
      scripts=[“bin/hello”,
               “bin/hellod”],
      data_files=[('/etc/cron.d', [‘etc/hello-cleanup']),
                  ('/etc/init.d', ['etc/hellod']),
                  ('/etc/httpd/conf.d', [‘httpd/hello.conf’]),
                  (‘/usr/share/hello’, [‘wsgi/hello.py’])])
```

RPM versions

- RPM versions have two components:
  - Version (which generally matches source)
  - Release (incremented when source does not change, but package does)
- Split on alpha/numeric boundaries and punctuation, sorted lexically (alpha) or numerically
- Working toward 1.0? Use 0.99.1, 0.99.2, …

setup.py → RPM versions

- RPM version number comes from the version parameter to setup()
- Release must be specified in setup.cfg:

```
[bdist_rpm]
release = 2
```

Dependencies

- Build requirements

```
[bdist_rpm]
build_requires = goodbye
                 sayonara
```

- Install requirements

```
[bdist_rpm]
requires = aloha > 0.9
           konichiwa
```

Scriptlets

- In setup.cfg, [bdist_rpm]
  - %install → install_script = …
  - %pre, %post → {pre,post}-install = …
  - %preun, %postun → {pre,post}-uninstall = …
- Install in particular is very useful…

Install scriptlet example

```
#!/bin/sh

# default action
python setup.py install --root=$RPM_BUILD_ROOT --record=INSTALLED_FILES

# purge source files; from http://www.pwan.org/wp/?p=47
find $RPM_BUILD_ROOT -name \*.py -exec rm {} \;
sed -i '/\.py$/d' INSTALLED_FILES

# set permissions on specific installed files
cat <<EOT >>INSTALLED_FILES
%attr(750,someuser,somegroup) /var/lib/hello/config
%attr(640,someuser,somegroup) %config(noreplace) /var/lib/hello/config/config.yaml
EOT
```

Source RPMs revisited

- Source RPMs can still be useful
  - Push to build infrastructure
  - Use with mock
  - Cross-compile i386 on x86_64
- `python setup.py bdist_rpm \`
  `--source-only`

Not enough?

- Have bdist_rpm make you a .spec file and make *that* your master

- `python setup.py bdist_rpm \`
  `--spec-only`

- Just remember to keep it in sync with setup.py changes…

So, why would you do this?

- Leverage existing RPM infrastructure
  - Deployment tools
  - Update from your own repository
- Because you're already doing non-Python packages

Questions?

(not that I know much more than I've presented…)

Further reading

- My RPM talk at DevOps West Michigan https://www.zigg.com/2014/slides-rpm-talk-devops-west-michigan.html

- Fedora packaging guidelines https://fedoraproject.org/wiki/Packaging:Guidelines

- Maximum RPM (dated, but still useful) http://www.rpm.org/max-rpm/