Configuring and Installing Python Packages the GNU Way With pyconfigure

Brandon Invergo

[2013-08-24 Sat]

Outline

GNU Coding Standards 7.1: Configuration

> *Each GNU distribution should come with a shell script named ‘configure’. This script is given arguments which describe the kind of machine and system you want to compile the program for. The ‘configure’ script must record the configuration options so that they affect compilation.*

https://gnu.org/prep/standards/

Configuration is a chore

No one wants to write a configure script by hand.
Hence: Autoconf
Related: writing robust Makefiles is an art.
Hence: Automake

GNU Coding Standards 3.1: Which Languages to Use

A recent change:

> *When highest efficiency is not required, other languages commonly used in the free software community, such as Scheme, Python, Ruby, and Java, are OK too.*

`https://gnu.org/prep/standards/`

The configuration standard and interpreted languages

The configuration tools are still largely C- and compilation-oriented. How should the configuration standard be implemented for these interpreted languages?

- Automake(?!) has a Python Autoconf macro
- autoconf-archive has some more Autoconf macros for other languages (including Python)

Many languages now have their own build systems, which is what people use instead of the Autotools.

Introducing: GNU pyconfigure

pyconfigure is a package containing handy Autoconf macros and template files for configuring and installing Python packages the GNU way

- For the developer: make standards-compliant Python packages
- For the user: provide the familiar `./configure && make install` installation process

Another Python installation system?

Python has several existing installation libraries:

- distutils
- distutils2
- setuputils
- bento

Standards aside, why bother?

On distutils and setuputils

"Setuputils monkey patches distutils. This has a serious consequence for people who have their own distutils extensions."

"distutils is not designed to be extensible... A couple of years ago, I decided that I could not put up with numpy.distutils extensions... anymore"

David Cournapeau, Numpy/Scipy & bento developer

`https://cournape.wordpress.com/2012/06/23/why-setuptools-does-not-matter-to-me/`

On bento

Let's look at its usage:

Example

```
bentomaker configure
bentomaker install
```

Why reinvent the wheel?

pyconfigure advantages

pyconfigure is based on Autoconf and Make, which means it's:

- robust & extendable
- language-agnostic
- familiar
- compliant with the GNU standards
- for the user: no new unusual dependencies

So, how is it used?

Choose your "back-end"

- Make: write your installation logic in Make, add setup.py as a wrapper around it
- distutils: write your installation logic in Python, add Makefile as a wrapper around it

Write your PKG-INFO file

Contains basic project metadata. Standardized in PEP 345.

Example (Minimal PKG-INFO example)

```
Metadata-Version: 1.2
Name: foo
Version: 1.5
Author-email: bug-foo@gnu.org
```

`http://www.python.org/dev/peps/pep-0345/`

Copy/generate the template files

From the project directory: `pyconf [TARGET] PKG-INFO`
TARGET is 'distutils' by default (the only current option)
Options:

- `-m, --prefer-make`: use the Make back-end
- `--no-make`: do not generate a Makefile
- `--macros-only`: only copy the Autoconf macros
- `--overwrite`: overwrite existing files

configure.ac

Example (A minimal configure.ac)

```
AC_INIT([foo], [1.5], [bug-foo@gnu.org])
m4_include([m4/python.m4])
PC_INIT([2.7], [3.3.2])
AC_CONFIG_FILES([Makefile setup.py])
AC_OUTPUT
```

Autoconf macros: finding the interpreter

- `PC_INIT([MIN-VERSION], [MAX-VERSION])` *(convenience macro)*
- `PC_PROG_PYTHON([PROG-TO-CHECK-FOR], [MIN-VERSION], [MAX-VERSION])`
- `PC_PROG_PYTHON_CONFIG([PROG-TO-CHECK-FOR])`
- `PC_PYTHON_VERIFY_VERSION([OPERATOR], [VERSION], [ACTION-IF-TRUE], [ACTION-IF-FALSE])`

Autoconf macros: getting system information

- `PC_PYTHON_CHECK_VERSION`
- `PC_PYTHON_CHECK_PLATFORM`

Autoconf macros: finding locations

- PC_PYTHON_CHECK_PREFIX
- PC_PYTHON_CHECK_EXEC_PREFIX
- PC_PYTHON_CHECK_SITE_DIR
- PC_PYTHON_SITE_PACKAGE_DIR
- PC_PYTHON_CHECK_EXEC_DIR
- PC_PYTHON_CHECK_EXEC_PACKAGE_DIR

Autoconf macros: developing extensions in C

- `PC_PYTHON_CHECK_INCLUDES` *(check the include flags for the Python headers)*
- `PC_PYTHON_CHECK_CFLAGS`
- `PC_PYTHON_CHECK_LDLAGS`
- `PC_PYTHON_CHECK_LIBS` *(check the LIBS flags)*
- `PC_PYTHON_CHECK_HEADERS` *(find the location of the Python headers)*
- `PC_PYTHON_TEST_LIBS` *(test for the presence of the Python shared libraries)*
- `PC_PYTHON_CHECK_EXTENSION_SUFFIX` *(Python 3 only)*
- `PC_PYTHON_CHECK_ABI_FLAGS` *(Python 3 only)*

Autoconf macros: performing tests

- PC_PYTHON_CHECK_MODULE
- PC_PYTHON_CHECK_FUNC
- Plus write test programs in Python via AC_LANG_PROGRAM:

Example (Writing test programs)

```
AC_LANG_PUSH(Python)[]
AC_RUN_IFELSE([AC_LANG_PROGRAM([dnl
# some code here
import foo
], [dnl
    # some more code here
    foo.bar()
])], [ACTION-IF-SUCCESSFUL], [ACTION-IF-FAILED])
AC_LANG_POP(Python)[]
```

Makefile.in (Make back-end)

Basic Python-based packages can be easily installed with Make via the customization of some variables:

- `PYPACKAGES`: Python modules to be installed (in the site-packages dir)
- `PYPACKAGE_ROOT`: The location of the modules (i.e. the `src` subdir)
- `SCRIPTS`: Scripts to be installed
- `PKG_DATA`: Data files to be installed in the site-packages dir
- `DATA`: Data files to be installed to datadir (i.e. `/usr/share/foo`)
- `DATA_ROOT`: The location of data files in the source directory
- `DISTFILES`: Files to be included in distributions of the package

But of course, it's greatly expandable. . .

setup.py (distutils back-end)

See the distutils documentation. . . .
But remember that this file can be configured via the configure script!

Future directions

What's coming up:

- Automake integration
- New back-ends: setuptools, bento, distutils2

Fin

Thanks!

`http://www.gnu.org/software/pyconfigure`

`https://ftpmirror.gnu.org/gnu/pyconfigure`

Version 0.2.1 released yesterday. Get it while it's hot.