shrinkwrap Documentation

Release 0.9

Stan Seibert

September 20, 2012

CONTENTS

Shrinkwrap provides tools to create lightweight Python package wrappers around non-Python software, and to install such software from project-specific repositories using virtualenv and pip.

A shrinkwrap package is a minimal python package that downloads, compiles and installs software to the virtualenv base directory. The shrinkwrap package behaves just like a regular python package, so both shrinkwrap and non-shrinkwrap packages can be dependencies of each other. We find that this greatly simplifies deployment of Python packages that depend on compiled libraries *without* requiring the use of system-wide packaging tools, like apt or yum. Shrinkwrap is **not** an API wrapper generator like SWIG, but does make it easier to install C libraries into a virtualenv before installing a separate Python wrapper around its API.

> **Warning:** To avoid cluttering PyPI with non-Python software, please do not ever upload shrinkwrap-generated packages there! Hosting shrinkwrap packages yourself is easy, and described further in *Running your own package index*.

Shrinkwrap is available on PyPI and can be installed via `pip install shrinkwrap`, `easy_install shrinkwrap`, or by downloading the package and installing with `python setup.py install`.

For those interested in the newest features should use the development version of shrinkwrap, available on bitbucket:

```
hg clone http://bitbucket.org/seibert/shrinkwrap
```

Note that you cannot directly install shrinkwrap from the Mercurial repository to a virtualenv with the `pip -e` command, as this appears to bypass our post- installation tasks. Running `python setup.py install` from the cloned repository is fine, however.

CHAPTER

ONE

CONTENTS

1.1 How Shrinkwrap Works

Shrinkwrap was designed to scratch a particular itch: automating the installation of software in a self-contained environment.

Nearly all package managers assume they are managing software installation for the entire system, and don't allow isolated environments. ZeroInstall is a notable exception which installs software into a user's home directory and can manage multiple versions of a single package. However, ZeroInstall is more aimed at providing specific applications to an end user, rather than providing a shell environment with a set of packages installed and ready for use.

Python's virtualenv and easy_install/pip tools are a great example of a userspace packaging solution that creates self-contained, easy-to-use environments. Since we work in mixed environments where C, C++ and Python software needs to coexist, shrinkwrap is our attempt to bring all that software under one roof.

Shrinkwrap achieves this goal by modifying the virtualenv in three ways:

1. It provides an alternate implementation of the `python setup.py install` command that makes it easy to insert an arbitrary installer function. This function typically downloads source tarballs, unpacks them, compiles and installs to the virtualenv base directory.

2. It adds new optional keyword arguments to `setup()` to specify the installer function, and also to list packages that are build dependencies of this package. See *Package Dependencies* for details.

3. It creates a new `$VIRTUAL_ENV/env.d` directory and patches the `$VIRTUAL_ENV/bin/activate` script to source any files found in that directory. This allows shrinkwrap packages to install scripts that make changes to the shell environment. Many of the software packages we work with require environment variables to function properly.

With these changes, it becomes very easy to write small Python packages that download, compile, and install software into the virtualenv environment.

1.1.1 Installing shrinkwrap packages

Installing a shrinkwrapped package is identical to installing any package with pip. You can install the package file directly from the filesystem:

```
pip install curl-7.27.0.tar.gz
```

or from a URL:

```
pip install http://mtrr.org/packages/curl/curl-7.27.0.tar.gz
```

or set the URL of extra package repository which contains shrinkwrapped packages:

```
export PIP_EXTRA_INDEX_URL=http://mtrr.org/packages/
pip install curl
```

If the shrinkwrapped package contains the standard test for the shrinkwrap module at the top (see *Built-in installers*), then shrinkwrap will be automatically installed to your virtualenv.

Note that environment files in the `$VIRTUAL_ENV/env.d/` directory are only sourced when the `$VIRTUAL_ENV/bin/activate` script is sourced. You will need to source that script again to refresh your environment if the shrinkwrapped package added new environment files.

As true Python packages, shrinkwrapped packages can be used in a requirements.txt files passed to pip. Just place the extra index URL argument at the top:

```
--extra-index-url http://mtrr.org/packages/
curl
fftw
nose
```

1.1.2 Limitations

We are the first to admit that shrinkwrap is straining the intended purpose of the Python packaging tools. As a result, shrinkwrap+pip has some shortcomings compared to more sophisticated package managers:

- Shrinkwrap can only be used with a virtualenv environment. Shrinkwrap tests for the presence of `$VIRTUAL_ENV` and will throw an exception if it is not found.
- Shrinkwrapped packages have only been tested to work with `pip` as the installer. We do not support using `easy_install`, although it might work.
- `pip uninstall` does not remove files installed by the package because we do not yet have a way to track changes made by arbitrary installer functions to the filesystem.
- We do not provide any mechanism to have build options selected at installation time (such as variants in MacPorts or USE variables in Gentoo emerge). Repositories of shrinkwrapped packages are easy to make with the `shrinkwrap` command line tool, so we encourage you to have different repositories for different kinds of deployment environments. Then you can tailor the build options of your source packages for each one.
- Shrinkwrap is currently focused on deploying source packages rather than precompiled packages. There is no reason you can't unpack a tarball of compiled code in a shrinkwrap `installer()` function, but we do not provide any mechanism for having packages compiled for different architectures in the same repository. Again, repositories are easy to make, so we would suggest one per architecture if this is your use case. (Selecting platform-specific build options in your `installer()` function, however, is no problem.)

1.2 Creating Packages

1.2.1 Writing wrapper packages

The goal of shrinkwrap is to create Python packages for non-Python software and install them using Python package management. To this end, shrinkwrap "packages" are simply setuptools setup scripts which know how to download and install other software.

A shrinkwrap package may either use a built-in installer function (for common installation methods) or define a custom one.

Built-in installers

An installer for software using autoconf is available to simplify things. The autoconf shrinkwrap installer assumes the tarfile unpacks to a directory with the same name as the tarfile with .tar.{gz,bz2} removed, contains a configure script, and understands the –prefix option to control where the package is installed.

For example, to package *curl*, one may write the following (as, e.g. *curl-7.27.0.py*):

```
try:
    from shrinkwrap.install import ShrinkwrapInstall
except ImportError:
    import subprocess; subprocess.check_call('pip install -b $VIRTUAL_ENV/build/build-shrinkwrap shri
    from shrinkwrap.install import ShrinkwrapInstall
from setuptools import setup


version = '7.27.0'

setup(
    name='curl',
    version=version,
    author='Stan Seibert',
    author_email='stan@mtrr.org',
    shrinkwrap_installer='autoconf',
    shrinkwrap_source_url='http://curl.haxx.se/download/curl-%s.tar.bz2' % version,
    cmdclass={'install': ShrinkwrapInstall},
)
```

To install curl, one would simply run:

```
python curl-7.27.0.py install
```

The boilerplate at the top of the script is required to ensure that shrinkwrap is installed before setuptools is imported. The `cmdclass` option must be set as shown above in order to use the `ShrinkwrapInstall` command rather than the default install command provided by setuptools.

The filename of the python script is not used by shrinkwrap.

Custom installers

The following package for bzip2 illustrates a custom installer function:

```
try:
    from shrinkwrap.install import ShrinkwrapInstall
except ImportError:
    import subprocess; subprocess.check_call('pip install shrinkwrap', shell=True)
    from shrinkwrap.install import ShrinkwrapInstall
import os
from setuptools import setup


version = '1.0.6'
source_url = 'http://www.bzip.org/%(version)s/bzip2-%(version)s.tar.gz' % {'version': version}


def installer(self):
    self.download_and_unpack_tarball(source_url)

    bzip2_dir = 'bzip2-' + version
```

```
    os.chdir(bzip2_dir)
    self.make(extra_opts=['install', 'PREFIX=%s' % self.virtualenv])

setup(
    name='bzip2',
    version=version,
    author='Andy Mastbaum',
    author_email='mastbaum@hep.upenn.edu',
    shrinkwrap_installer=installer,
    cmdclass={'install': ShrinkwrapInstall},
)
```

As before, installing bzip2 simply requires:

```
python bzip2-1.0.6.py install
```

Here, the `shrinkwrap_installer` argument to `setup()` is set to our own installer function. The installer uses two shrinkwrap-provided convenience functions, `download_and_unpack_tarball` and `make` to download, untar, and compile the bzip2 library. See *shrinkwrap.install* for a complete list of convenice functions. By passing extra options to make, the software is installed into the root of the active virtualenv.

Note: For several examples of custom installers, see https://bitbucket.org/seibert/shrinkwrap_pkgs.

Examples include getting code from version control, installing with cmake, and customizing install paths.

Package Dependencies

The handling of package dependencies in pip (as of version 1.1, anyway) does not, unfortunately, meet the requirements for installing compiled software. Packages listed in the `install_requires` keyword argument to `setup()` will be discovered by pip and installed, but in an arbitrary order. Packages listed in the `setup_requires` keyword argument are not actually installed to the virtualenv, but rather made available in .egg files in the build directory, which also is not a solution for compiled code.

As a result, shrinkwrap adds a new keyword argument, `shrinkwrap_requires`, to `setup()`. All dependencies listed here are guaranteed to be fully installed to the virtualenv before the `installer()` function is run. A separate pip process is spawned to install each one, so the dependency string can include package versions, just as `setup_requires` and `install_requires` allow. The shrinkwrap dependencies can be any kind of package, so if your library uses SCons as the build system, you can list it in the `shrinkwrap_requires` field:

```
setup(
    name='foo',
    version=version,
    author='Example Author'
    author_email='author@example.com'
    shrinkwrap_installer=installer,
    shrinkwrap_requires=['scons'],
    cmdclass={'install': ShrinkwrapInstall},
)
```

Environment Variables

If your package also needs to set some shell environment variables for operation, they can be placed in the `$VIRTUAL_ENV/env.d` directory, and they will be sourced automatically when the `$VIRTUAL_ENV/bin/activate` script is sourced. See the package file for CERN's ROOT library for an example of using the `install_env()` convenience function.

1.2.2 Packaging for a package index

To share shrinkwrap packages via a package index like PyPI (remember: don't actually upload shrinkwrap packages to PyPI), you must create distribution tarballs. Shrinkwrap includes a tool to create these automatically from wrapper setup.py files:

```
shrinkwrap createpkg bzip2-1.0.6.py
```

This will create a tarball in the current directory suitable for uploading. Wildcards are valid for generating many packages at once.

Note: You can name the python file anything you want (not just setup.py). It will be copied to setup.py in a temp directory and the name of the resulting tarball will be derived from the package metadata you specified.

1.2.3 Running your own package index

As mentioned in the *Limitations* section, shrinkwrap packages are small and easy to deploy. It is better to have different package repositories when you want to build a source package with different options in different situations, rather than have One Repository To Rule Them All [1] .

If you wish to serve your own package index, use the `-p` option to place output tarballs into one properly-formatted directory:

```
shrinkwrap createpkg -p packages my_shrinkwrap_pkgs/*
```

Simply serve the *packages* directory on the web, and pip clients can interact with it just like PyPI:

```
client$ export PIP_EXTRA_INDEX_URL=http://your-server.com/packages/
client$ pip install bzip2
```

Extra package indices can be specified in a requirements.txt file with an option at the top:

```
--extra-index-url http://mtrr.org/packages/
```

1.3 API Documentation

1.3.1 `shrinkwrap.install`

utilities for installing shrinkwrapped packages

class `shrinkwrap.install.`**`ShrinkwrapInstall`**(*dist*)

Base class for a setup.py "install" command that wraps a generic tarball installation.

`download_and_unpack_tarball`(*url*, *to_src_dir=False*)

Convenience method that downloads a tarball from the given URL and unpacks it. By default, the tarfile and uncompressed contents are placed in the current directory, but if to_src_dir is True, then both the tarfile and contents will be in the src/ directory under the virtualenv base.

Returns the full path to the downloaded tar file.

[1] In order to avoid creating One Repository To Rule Them All, we do not plan to ever offer a repository of "official" shrinkwrap packages. Feel free to copy other people's package files to your repository, however.

download_url (*url*, *saveto=None*)
Downloads a file. If no saveto is specified, the basename of the URL and the current directory will be used.

Returns the full path to the saved file on disk.

env_dir
Full path to shrinkwrap env directory inside virtualenv

install_env (*filename*, *contents*)
Create an environment shell script called `filename` in the shrinkwrap env directory and fill it with the string `contents`.

make (*parallel=True*, *extra_opts=None*)
Run make in current directory. If `parallel` is true, will run make with the -j option and the number of CPU cores. `extra_opts` is a list of additional options to be passed to make.

Remember to escape them for the shell, if needed!

ncpu
Number of CPU cores.

python_libdir
Location of python library.

run ()
setuptools install command that install dependencies in before calling the installer function provided in the shrinkwrap_installer keyword argument to setup().

shell (*cmd*, *success_return_code=0*)
Runs `cmd` in a shell. Raises subprocess.CalledProcessError if the return code from `cmd` is not equal to `success_return_code`.

src_dir
Full path to source directory inside virtualenv

untar (*source*, *target='.'*, *makedir=False*)
Unpack tar file with filename `source` to directory `target`. If `makedir` is true, then the target directory will be created first, including missing parent directories.

Default is to extract to current directory.

virtualenv
Full path to base of virtualenv directory.

shrinkwrap.install.**assert_string** (*dist*, *attr*, *value*)
Verify that value is a string

shrinkwrap.install.**autoconf_install** (*self*)
A convenience function to perform an autoconf-based installation.

Note: requires parameter "shrinkwrap_source_dir" to be set in setup() call.

shrinkwrap.install.**validate_installer_option** (*dist*, *attr*, *value*)
Verify that value is either an allowed string or a callable function

1.3.2 shrinkwrap.command

shrinkwrap command-line utility

shrinkwrap.command.**activate** ()
Print the contents of scripts in `$VIRTUAL_ENV/env.d`.

shrinkwrap.command.**createpkg**(*argv*)
Take each listed file on the command line, copy to a temporary directory, rename to setup.py, run "python setup.py sdist", and copy back the generated tar file.

Streamlines the creation of shrinkwrap packages which are entirely defined by a setup.py file.

shrinkwrap.command.**help**()
Print usage for the shrinkwrap command-line utility.

1.4 Roadmap

Near future goals:

- Automated testing suite. Due to the nature of shrinkwrap, this will require the testing framework to repeatedly create and destroy entire virtualenvs for each test, which requires some non-trivial infrastructure coding.
- SHA1 hash checking of downloaded tar files.
- An interface for shrinkwrap packages to test and report whether the wrapped software is already installed system-wide. This would allow a shrinkwrap package to optionally skip a lengthy compile and install step when the appropriate system packages are already present.

Far future goals:

- Remove all installed files when pip uninstall is called. This will require detecting and recording all the files added to the virtualenv by the installer() function.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

S