The Black Magic of Python Wheels

Elana Hashman
Python Packaging Authority
PyCon US 2019 – Cleveland, OH
@ehashdn

Wheels/Black Magic FAQ

Q: But I’m not a witch?!

A: *Sometimes the greater good requires a little sacrifice.*

Topics

- Python packaging and distribution
- ELF (Executable and Linkable Format) files
- Dynamic linking
- ABIs (Application Binary Interfaces) and symbol versioning

Outline

- A brief history of Python packaging and distribution
- An overview of the wheel
- Why we need native extensions
- How do native extensions even work, really?
  - What are `manylinux` and `auditwheel` for?
- How you can get involved

A Brief History of Python Packaging: Eggs

- Organically adopted (no guiding PEP)
- No standard → many incompatible implementations
- Designed to be directly importable, could include compiled Python (`.pyc` files)

Python Packaging Reinvented: The Wheel

- Adopted via PEP 427
- Follows the PEP 376 standard for distributions and PEP 426 standard for package metadata
- Designed for distribution, cannot include `.pyc` files (but may include other pre-compiled resources)

Wheels “make it easier to roll out” Python

- Pure wheels
  - Only contain Python code
  - May target a specific version of Python
- Universal wheels
  - Python 2/3 compatible pure wheels

```
pip install wheel
python setup.py bdist_wheel
```

Wheels “make it easier to roll out” Python

- Pure wheels
  - Only contain Python code
  - May target a specific version of Python
- Universal wheels
  - Python 2/3 compatible pure wheels
- Extension wheels
  - ???

Extensions without binary distributions

```
$ pip install cryptography  # source-only download

...
    c/_cffi_backend.c:2:10: fatal error: Python.h: No
such file or directory
     #include <Python.h>
              ^~~~~~~~~~
    compilation terminated.
    error: command 'x86_64-linux-gnu-gcc' failed with
exit status 1

$ sudo apt install python-dev  # get Python.h
```

Extensions without binary distributions

```
$ pip install cryptography

...
    c/_cffi_backend.c:15:10: fatal error: ffi.h: No
such file or directory
     #include <ffi.h>
              ^~~~~~~
    compilation terminated.
    error: command 'x86_64-linux-gnu-gcc' failed with
exit status 1

$ sudo apt install libffi-dev  # get ffi.h
```

Extensions without binary distributions

```
$ pip install cryptography

...
   build/temp.linux-x86_64-2.7/_openssl.c:498:10: fatal
error: openssl/opensslv.h: No such file or directory
     #include <openssl/opensslv.h>
              ^~~~~~~~~~~~~~~~~~~
    compilation terminated.
    error: command 'x86_64-linux-gnu-gcc' failed with
exit status 1

$ sudo apt install libssl-dev  # get opensslv.h
```

Extensions without binary distributions

```
$ time pip install cryptography

Successfully installed asn1crypto-0.24.0 cffi-
1.11.5 cryptography-2.3.1 enum34-1.1.6 idna-2.7
ipaddress-1.0.22 pycparser-2.19 six-1.11.0


real 0m16.369s
user 0m15.823s
sys  0m0.627s
```

Extensions with binary distributions

```
$ time pip install cryptography  # prebuilt binary

Successfully installed asn1crypto-0.24.0 cffi-
1.11.5 cryptography-2.3.1 enum34-1.1.6 idna-2.7
ipaddress-1.0.22 pycparser-2.19 six-1.11.0


real 0m1.088s
user 0m0.980s
sys  0m0.108s
```

What sort of black magic is this? ✨🔮

Extension Wheels are safe to `pip install`!

- Installing Python native extensions without wheels is painful
- Conda was developed to address this gap: why not use that?
  - Like eggs, Conda was not adopted by a PEP
  - Conda packages are not Python-specific, not supported by PyPI
  - Conda packages are not compatible with non-Conda environments
- Wheels are compatible with the entire Python ecosystem

What is a Python (Native) Extension?

- **Native:** the code was compiled specifically for my operating system
- **Extension:** this library extends Python's functionality with non-Python code
- **Example:** `cryptography`
  - It uses CFFI: the "C Foreign Function Interface" for Python

Python code is not just Python.

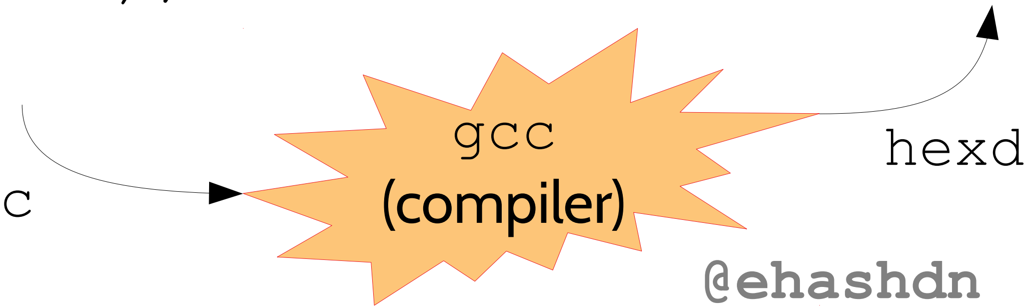
For Python to harness its full potential,
it must be able to depend on C libraries.

C is a compiled language

```
// hello.c

#include<stdio.h>

int main(void) {
   puts("hello
          world");
}
```

```
# a.out (hexadecimal)

0000000 7f45 4c46 0201 0100
0000008 0000 0000 0000 0000
0000010 0300 3e00 0100 0000
0000018 5005 0000 0000 0000
0000020 4000 0000 0000 0000
...
```

ELF File



gcc hello.c

gcc
(compiler)

hexdump a.out

Hexes and ELFs

```
$ readelf -a a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00
           00 00 00 00 00 00 00 00
  Class:             ELF64
  Data:              2's complement, little endian
  Version:           1 (current)
  OS/ABI:            UNIX - System V
  ABI Version:       0
  Type:              DYN (Shared object file)
  Machine:           Advanced Micro Devices X86-64
...
```

Hexes and ELFs

```
$ readelf -a a.out
...
Program Headers:
  Type      Offset             VirtAddr           PhysAddr
            FileSiz            MemSiz              Flags  Align
  INTERP    0x0000000000000238 0x0000000000000238 0x0000000000000238
            0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

ELF interpreter

Hexes and ELFs

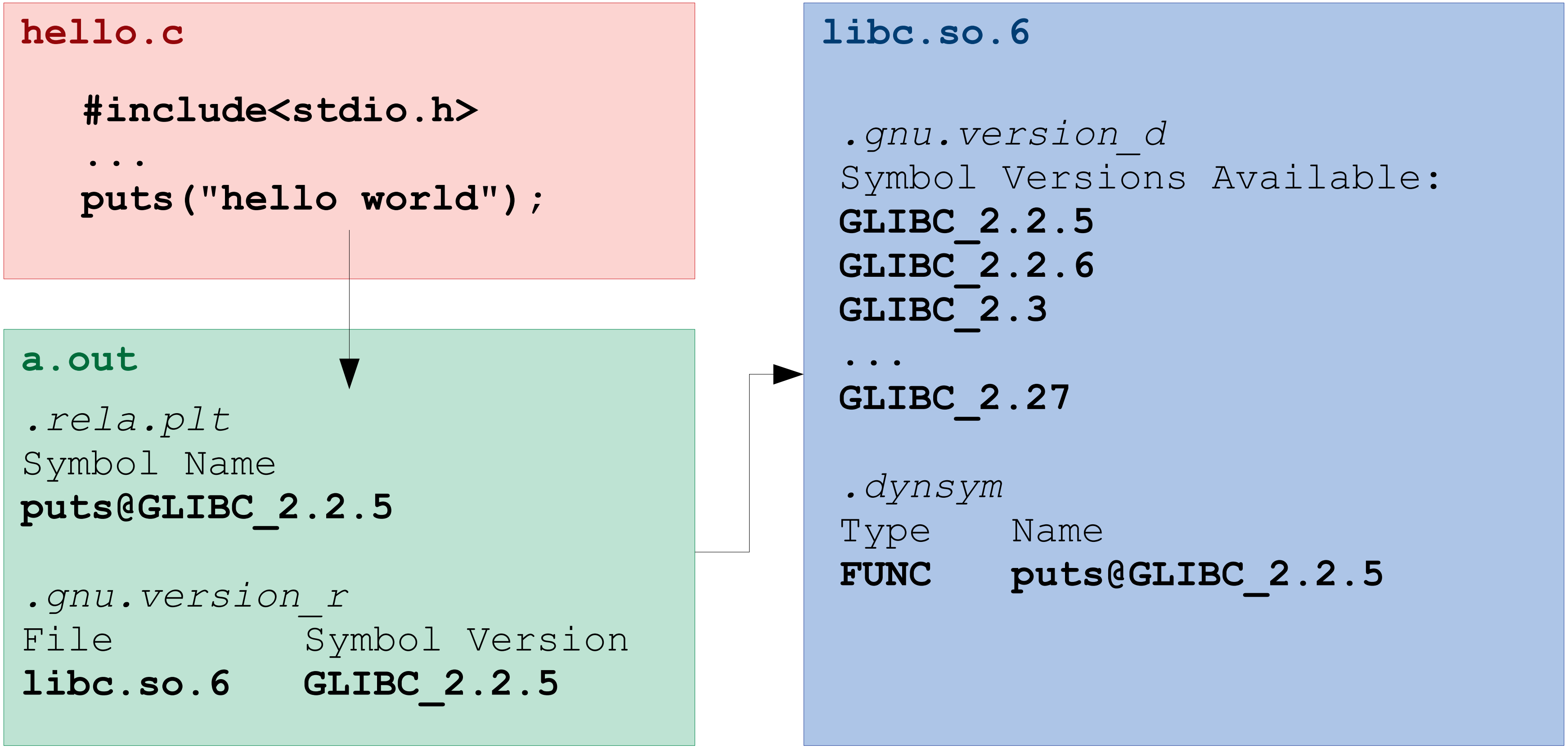
```
$ readelf -a a.out

...

Relocation section '.rela.plt' at offset
0x4d0 contains 1 entry:

  Offset          Info           Type
000000200fd0  000200000007 R_X86_64_JUMP_SLO

  Sym. Value     Sym. Name + Addend
0000000000000000 puts@GLIBC_2.2.5 + 0
```

Symbol Version

Hexes and ELFs

```
$ readelf -a a.out
...
Version needs section '.gnu.version_r' contains
1 entry:
 Addr: 0x00000000000003f0  Offset: 0x0003f0  Link: 6
(.dynstr)
  000000: Version: 1  File: libc.so.6  Cnt: 1
  0x0010:   Name: GLIBC_2.2.5  Flags: none  Version: 2
```

Symbol Versions in Action



hello.c

```
#include<stdio.h>
...
puts("hello world");
```

a.out

.rela.plt

```
Symbol Name
puts@GLIBC_2.2.5
```

.gnu.version_r

```
File          Symbol Version
libc.so.6     GLIBC_2.2.5
```

libc.so.6

.gnu.version_d

```
Symbol Versions Available:
GLIBC_2.2.5
GLIBC_2.2.6
GLIBC_2.3
...
GLIBC_2.27
```

.dynsym

```
Type    Name
FUNC    puts@GLIBC_2.2.5
```

What happens when we run this?

- OS parses “magic ELF” text
- OS invokes the ELF interpreter specified by the binary
- ELF interpreter loads any required files with valid versions
- ELF interpreter relocates the program code and dependencies in memory so that it can run
- This is called *dynamic linking*

How to get C into 🐍??

- **Old way:** make users compile from source
  - Obtaining dependencies is the user's problem
  - Compile against system library version at Python install time
- **New way:** users install pre-built Python wheels
  - Bundle pre-compiled binary dependencies inside a Python wheel ✨

How to get C into 🐍??

- The old ways have many problems
  - Slow (compiling from source)
  - Version mismatches
  - Requires knowledge of system package management
- Python wheels solve this!
  - Dependencies provided are the right versions and precompiled
  - Wheels are Python-native: just `pip install` them

But how can we ensure the pre-compiled binaries are compatible with *my* system?

Q: How can we ship compiled Python extensions compatible with as many systems as possible?

A: Symbol versioning (`manylinux`)
and dependency bundling (`auditwheel`).

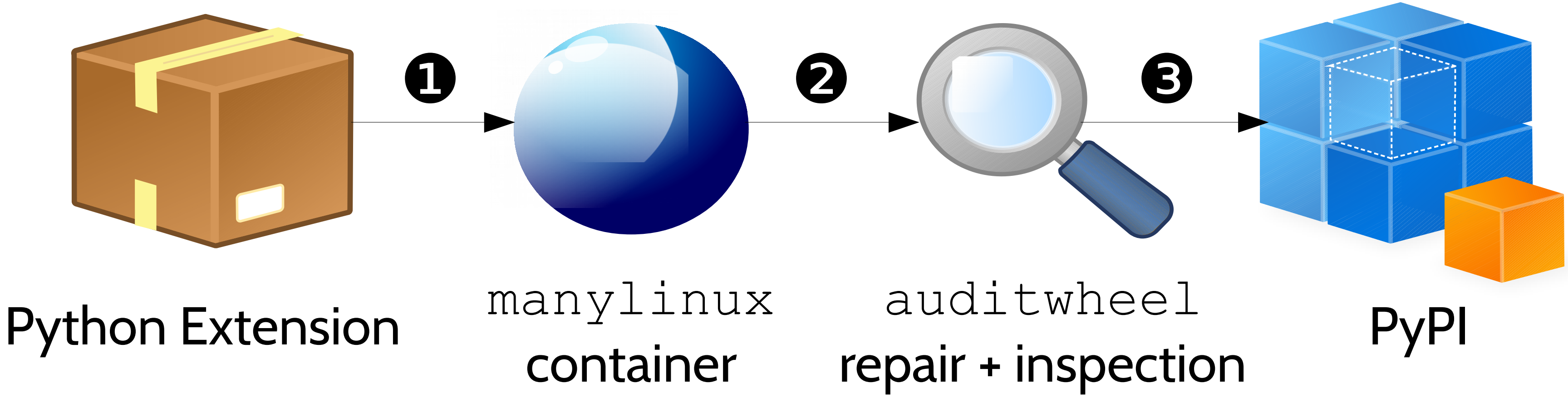
What are `manylinux` and `auditwheel`?

- PEPs 513 and 571 define a set of permitted libraries and their symbol versions for Linux systems
  - "Many" Linux systems are compatible with this standard
- `manylinux` is both the name of the policy and a Docker image
  - `manylinux1` (PEP 513): CentOS 5, i386/amd64 architectures
  - `manylinux2010` (PEP 571): CentOS 6, i386/amd64 architectures
- `auditwheel` is a tool to enforce the symbol policies

What is `auditwheel`?

- `auditwheel` uses dark magic to vendor external binary dependencies into your wheel 👻
- Empowers developers to build `manylinux` wheels without having to change their build processes
  - Just build and install dependencies in the `manylinux` image
  - `auditwheel repair` will bundle dependencies into a compliant wheel

Wheel Builder's Pipeline for Linux



❶ Add your code, dependencies to the `manylinux` Docker image and build against your supported Python versions/architectures

❷ Repair and inspect the built wheel with `auditwheel` for compliance

❸ Upload to PyPI!

Want in on the magic?

- Help build wheels!
  - Feedback enthusiastically welcomed ✨
- pythonwheels.com
  - See what packages already build wheels
  - Find examples for how to build yours (including Windows, OS X)
- github.com/pypa/python-manylinux-demo
  - Simple demo to learn Linux wheelbuilding

`auditwheel` needs a new maintainer

- I'm stepping down after three years!
  - hashman.ca/leaving-pypa
- The `manylinux` Docker images and new `manylinux2014` spec need some love, too

Questions?

Thanks to:

Red Hat
Nelson Elhage, Paul Kehrer, Donald Stufft

Talk resources: https://hashman.ca/pycon-2019

Image License Information

- **Tree Cat Silhouette Sunset:** Public Domain (CC0) @besshamiti **https://plixs.com/photo/3297/tree-cat-silhouette-sunset**
- **Happy Halloween!** (Costume Dog): Public Domain (CC0) @milkyfactory **https://flic.kr/p/ArW1N9**