Contributing to Open Source Software

Joe LaFreniere (lafrenierejm)

2017-09-23

Outline

Why does this talk exist?

What is open source software?

What constitutes a contribution?

How is OSS published?

Where are contributions managed?

Notable OSS Examples

- Wikipedia, Unreal Engine 4, this presentation
- Firefox, Chromium
- VLC, LibreOffice
- Atom, vscode, Emacs, Vim
- Git, Subversion, 7zip
- OpenSSL, OpenGL
- Python, C, Swift
- Linux, *BSD

Open Source Initiative's Criteria for OSS

1. free redistribution
2. source code
3. derived works
4. integrity of author's source code
5. no discrimination against persons or groups
6. no discrimination against fields of endeavor
7. distribution of license
8. license must not be specific to a product
9. license must not restrict other software
10. license must be technology-neutral

Types of Contributions

- source code
- formal documentation
- testing
- bug reporting
- answering questions

Source Code — Technical

- follow code guidelines
  - formal and informal
  - editor settings
    - whitespace cleanup
    - tpope's sleuth.vim
- provide tests alongside code
  - never reduce coverage
- strive for atomic commits
- linter and tests

Source Code — Social

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Source Code — Social

- provide context for the change
  - reference issue tracker
- respond to questions and reviews
  - treat continuous integration as review
- keep commit message subject short
  - fewer than 50 chars
  - detail changes in message body

Testing

general

- language-specific tools
- multiple frameworks to choose from
- consistency
  - sandboxed environment
- coverage metrics
- UTD offers SE 4367

automated

- linters
  - *undesirable bits of fiber and fluff*
- continuous integration
  - Travis CI et al.
- self-testing build

Bug Reporting

- existing report?
- context for bug
  - if needed, justify expectation
- establish minimum working example (MWE)
- provide details upfront
  - software version
  - operating system
  - error messages
- ask for help obtaining details
- detail fix, mark resolved

Bug Reporting — MWE

Goals

- "what helps others understand my problem?"
- "what gets in the way of understanding my problem?"
- reproducability
  - side effectful -> explicitly list input

Faulhammer on TEX

- remove unnecessary packages
- built-in > user-defined
- strip unnecessary files
- provide needed files

Documentation — Reference Materials

18.1.2.2. Constants

The AF_* and SOCK_* constants are now `AddressFamily` and `SocketKind` `IntEnum` collections.

New in version 3.4.

socket.**AF_UNIX**
socket.**AF_INET**
socket.**AF_INET6**

These constants represent the address (and protocol) families, used for the first argument to `socket()`. If the `AF_UNIX` constant is not defined then this protocol is unsupported. More constants may be available depending on the system.

socket.**SOCK_STREAM**
socket.**SOCK_DGRAM**
socket.**SOCK_RAW**
socket.**SOCK_RDM**
socket.**SOCK_SEQPACKET**

These constants represent the socket types, used for the second argument to `socket()`. More constants may be available depending on the system. (Only `SOCK_STREAM` and `SOCK_DGRAM` appear to be generally useful.)

socket.**SOCK_CLOEXEC**
socket.**SOCK_NONBLOCK**

These two constants, if defined, can be combined with the socket types and allow you to set some flags atomically (thus avoiding possible race conditions and the need for separate calls).

Figure: Excerpt of `docs.python.org/3/library/socket`

Documentation — Reference Materials

- developer-facing
- consulted in place of source
- requires knowledge of source implementation
- inconsistency with source = undocumented behavior
- if changing code, insist on doc change
  - mark with “NOMERGE” tag in meantime
  - totally valid to request additional help

Documentation — Examples

Example:

```
>>> import socket
>>> s1, s2 = socket.socketpair()
>>> b1 = bytearray(b'----')
>>> b2 = bytearray(b'0123456789')
>>> b3 = bytearray(b'--------------')
>>> s1.send(b'Mary had a little lamb')
22
>>> s2.recvmsg_into([b1, memoryview(b2)[2:9], b3])
(22, [], 0, None)
>>> [b1, b2, b3]
[bytearray(b'Mary'), bytearray(b'01 had a 9'), bytearray(b'little lamb---')]
```

Figure: Example from `docs.python.org/3/library/socket`

Documentation — Examples

- developer- and user-facing
- consulted in place of source or reference
- inconsistency with source = the *worst*
- prevent breakage of project-affiliated examples
  - notify third parties ahead of time
  - update QA answers upon publication

Documentation — Tutorials

9. Classes

Classes provide a means of bundling data and functionality together. Creating a new class creates a new *type* of object, allowing new *instances* of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

Figure: Excerpt of `docs.python.org/3/tutorial/classes`

Documentation — Tutorials

- user-facing
- inconsistency with source = loss of credit
- not feasible to keep up-to date indefinitely
  - specify versions, dates
- link to project documentation
- don't delete if outdated, just be upfront

Publishing OSS

Maintained by Project

- source code
  - available per definition of OSS
- binaries prebuilt by project
- package manager generally preferred

Package Management

- language
  - Python: pip, PyPi
  - Ruby: RubyGems
- operating system
  - Windows: Chocolatey
  - macOS: Homebrew
  - Linux: so many

Where are contributions managed?

- bug trackers
- forums, wikis, chat
- mailing lists
- code-hosting websites

Bug Trackers

Git{Hub,Lab} issues

Bugzilla

- most notably used by Mozilla for Firefox
- very formalized process

Jira, Trello

- widely used in private sector

Forum, Wiki, Chat (IRC/Slack/Gitter/...)

Mailing Lists

- plaintext only
  - adhere to text width
  - format=flowed
- attached vs. inline
- send to individual, CC list
- typically mirrored on website

Code-Hosting Websites and Pull Requests

- pull request workflow