Git Cola Python 3 Port Retrospective

A look back at how Git Cola transitioned to Python 3

What is Git Cola?

- Cross-platform user interface for Git
  - Linux / BSD
  - macOS
  - Windows
- Pure-Python application
  - PyQt interface
  - Git command-line backend
  - Native language translations for Chinese, Japanese, Spanish, Portuguese, and other languages
- Goal: maximum compatibility
  - PyQt 4 + 5
  - Python 2 + 3
  - Cross-platform (Windows in particular was the biggest effort)

Why Maximum Compatibility?

- Git Cola was first ported in 2014
- Desire to retain a single code base with no branches
- Python2 would continue to be in heavy use at the time (and still is)
- Wanted to avoid splitting the user base along dependency boundaries
  - Qt 4 vs. 5
  - Python 2 vs. 3
- Simplicity was ***not*** a motivation
  - Python2+3 is (slightly) more complex than a 2-only or 3-only code base

Pillars of Portable Python

- six, and it's okay to have your own "compat" module(s) providing more
- pylint --py3k
  - Use this **in addition** to your default pylint checks
  - They report different errors
- flake8 --format=pylint --doctests
- tox
- Get your CI to run pylint using python3 as well as python2
  - They report different errors

Port the tests first

- Getting tox up and running is a great first step.
- *python -m pytest --doctest-modules ...*
  - Sensible default test runner. Invoke it this way to allow pytest to be found through the *python* in your $PATH, e.g. in a virtualenv
- Once you have your tests running through tox you can test permutations
- Personal preference: Stick your workflows in a Makefile
  - Easy to debug issues locally
  - ***make test*** # fast unit tests
  - ***make check*** # slow pre-commit checks run by CI

tox.ini - https://github.com/git-cola/git-cola/tree/master/tox.ini

```
minversion = 1.8
envlist = py{27,34,35,36,37}

[testenv]
sitepackages = true
deps =
     -rrequirements/requirements.txt
     -rrequirements/requirements-dev.txt
whitelist_externals =
     make
commands =
     make test
     make flake8
```

.travis.yml - https://github.com/git-cola/git-cola/blob/master/.travis.yml

```
language: python
python:
  - "2.7"
  - "3.4"
#  We use apt to avoid having to build PyQt ourselves, so that means
#  we can only use the default python versions available in Ubuntu.
#  For Trusty, that's 2.7.6 and 3.4.3.
virtualenv:
  system_site_packages: true

addons:
  apt:
     packages:
     # Build dependencies, not needed at runtime
     - python-sphinx
     - python3-sphinx
     # Runtime dependencies
     - python-qt4
     - python3-pyqt5
```

```
install:
  # Test dependencies
  - make requirements-dev
  # Build translations
  - make all
  # Build documentation
  - make doc

script:
  # Run tests
  - make test V=2
  - pylint --version
  - make pylint
  - make pylint3k
  - flake8 --version
  - make flake8
```

Python Application Architecture Redux

- Unicode first - make sure you test handling of non-ASCII unicode data
- When in doubt, assume and use UTF-8.
- Unicode Sandwich
  - External inputs and outputs are where encoding to bytes happens
  - Application internals deal with unicode strings exclusively
  - PyQt assumes this approach - APIs return unicode strings in SIP API v2 mode
- For PyQt apps, use the “qtpy” library
  - Provides a compatibility shim between Qt4, Qt5, PySide2, etc.
  - “Qt.py” is a compatible alternative providing the same functionality
  - Both aim to make Qt4 behave like Qt5
  - SIP API v2

Unicode Sandwich

For Python2+3 applications, this means all source files ***must*** begin with this stanza

```
from __future__ import absolute_import, division, unicode_literals
```

unicode_literals has non-local effects because the return values from functions defined in a module can change from bytes/str to unicode, which can affect callers, so be aware of the implications. This is important first step gets python2 to behave more closely to python3.

Unicode Sandwich

- Data driven! Be aware of your data and its encodings.
- ASCII is almost never the right answer in 2019.
- UTF-8 should be your default choice.
- If our pipeline is all ASCII, do we care?
  - What about translations?
  - What about user's names?
  - What about emoji, copyright symbols, etc? Gotta have emoji
- External subprocess calls need encoded byte str arguments for py2, and unicode str for py3. Compat wrap it.
- File open/read/write all need wrappers to get Py2 to return/take unicode str

Annoyances

- Be prepared to silence pylint/flake8 warnings “from the future”
  - Tools assume a full port, not a 2+3 compatible approach, so some warnings are useless
- pylint is slow, and it’s now twice as slow because you have to run it twice
  - py3k mode + normal ⅔ mode report different errors -- single invocation would be nice
- Swig supports Unicode->Utf8 conversions for std::string in Python2 mode
- Boost::python does not (not sure about pybind11)
  - Causes friction when applications have a unicode core, requires compat wrappers
- You will become a better Python developer
  - Because you now know about all the weird details that changed between 2 and 3.