CI Best Practices

Code Review ¶

All patches that go into a project repo need to be code reviewed by someone other than the original author. Code review is a great way to both learn from others as well as improve code quality. Contribution to code review is highly recommended regardless of activity as a committer.

Below provides a simple checklist of common items that code reviewers should look out for (Patch submitters can use this to self-review as well to ensure that they are not hitting any of these):

General

- Does the Git commit message sufficiently describes the change? (Refer to: https://chris.beams.io/posts/git-commit/)
- Does the commit message have an 'Issue: <someissue>' in the footer and not in the subject line or body?
- Are there any typos?
- Are all code review comments addressed?
- Is the code rebased onto the latest HEAD of the branch?
- Does the code pull in any dependencies that might have license conflicts with this project's license?

Code

- Are imports alphabetical and sectioned off by stdlib, 3rdparty, and local?
- Are functions / methods organized alphabetically? (or categorized alphabetically)
- Does the change need unit tests? (Yes, it probably does!)
- Does the change need documentation?
- Does every function added have function docs? (javadoc, pydoc, whatever the language code docs is)
- Does it pass linting?
- Does the code cause backwards compatibility breakage? (If so it needs documentation)

Note

Refer to Google's blog (google-blog-code-health) on effective code review.

Generic Linting (Coala)

Coala is a great tool for linting all languages. The easiest way to run Coala is with python-tox and requires Python 3 installed on the system:

```
tox -ecoala
```

Running Coala without Tox can come in handy for executing Coala in interactive mode. In this case, install Coala in a Python viritualenv. Use virtualenvwrapper as it makes it simple to manage local virtual environments.

Requirements

- Python 3
- Python virtualenv
- Python virtualenvwrapper

Install Coala

Note

Some distros have a package called *coala* available but do not confuse this package with python-coala which is an entirely different piece of software.

Using virtualenv (assuming virtualenvwrapper is available), install Coala:

```
mkvirtualenv --python=/usr/bin/python3 coala
pip install coala coala-bears
coala --help
```

For future usage of an existing virtualenv, activate as follows:

```
# Re-activate Coala virtualenv
workon coala
# Run the coala command
coala --help
```

Set up Coala for a Project

Use python-tox to manage a Coala setup for any projects that require linting.

Requirements

- Python 3
- Python virtualenv
- Python Tox

Configure the project with a tox.ini and a .coafile file. Below are examples of .coafile and tox.ini as defined by lftools. Inside the tox.ini file the interesting bits are under [testenv:coala].

.coafile

```
[all]
ignore = .tox/**,
 .git/**,
 .gitignore,
 .gitreview,
 .gitmodules,
 node_modules/**

[all.Git]
bears = GitCommitBear
ignore_length_regex = Signed-off-by,
 Also-by,
 Co-authored-by,
 http://,
 https://

[all.Documentation]
bears = WriteGoodLintBear
files = docs/**/*.rst

[all.MarkDown]
bears = MarkdownBear,SpaceConsistencyBear,WriteGoodLintBear
files = **.md, **.markdown
use_spaces = true

[all.Python]
bears = BanditBear,
 PEP8Bear,
 PyCommentedCodeBear,
 PyDocStyleBear,
 PyFlakesBear,
 PyImportSortBear
files = *.py
```

tox.ini

```
[tox]
minversion = 1.6
envlist =
 check-best-practices,
 check-hooks,
 coala,
 docs,
 docs-linkcheck
skipsdist=true

[testenv:check-best-practices]
commands = python {toxinidir}/check-best-practices.py

[testenv:check-hooks]
deps = pre-commit
commands =
 pre-commit install
 pre-commit run --all-files

[testenv:coala]
basepython = python3
deps =
 coala
 coala-bears
 nodeenv
commands =
 nodeenv -p
 npm install --global remark-cli remark-lint write-good
 python3 -m nltk.downloader punkt maxent_treebank_pos_tagger averaged_perceptron_tagger
 coala --non-interactive

[testenv:docs]
deps = -rrequirements.txt
commands =
 sphinx-build -j auto -W -b html -n -W -d {envtmpdir}/doctrees ./docs/ {toxinidir}/docs/_build/html

[testenv:docs-linkcheck]
deps = -rrequirements.txt
commands = sphinx-build -j auto -W -b linkcheck -d {envtmpdir}/doctrees ./docs/ {toxinidir}/docs/_build/linkcheck
```

Jenkins Job Builder

Jenkins Job Builder Best Practices