

Open Source Development Course

Continuous Integration and Delivery

Vojtěch Trefný

vtrefny@redhat.com

26. 3. 2020

 twitter.com/vojtechtrefny

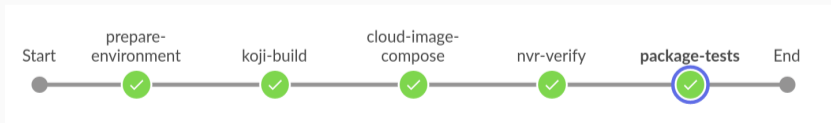
 github.com/vojtechtrefny

 gitlab.com/vtrefny

Pipeline

CI/CD Pipeline

- Steps that need to be performed to test and deliver new version of the software.
- Defines what needs to be done: when, how and in what order.
- Steps can vary for every project.
- Multiple pipelines or steps can run in parallel.



1. Testing environment

Preparation of the environment to run the tests: deploying containers, starting VMs...

2. Static Analysis

Finding defects by analyzing the code without running it.

3. Codestyle

Checking for violations of the language or project style guides.

4. Build

Building the project from source.

5. Tests























Running project test suite or test suites.

6. Packaging and Deployment

Building source archives, packages or container images.

Testing Environment

Testing Environment

Configuration Matrix	x86_64	i686	arm64
f_30	 	 	
f_31	 		 
f_rawhide	 		
centos_7	 		
debian_10	 	 	
debian_t	 		
rhel_8	 		

1. Preparation of VMs/containers to run the tests

We might want to run tests in different environments on multiple different distributions or architectures.

2. Installation of the test dependencies

Test dependencies are usually not covered by the project dependencies.

3. Getting the code

Clone the PR or get the latest code from the master branch.

Static Analysis

- Tools that can identify potential bugs by analyzing the code without running it.
- Can detect problems not covered by the test suite – corner cases, error paths etc.
 - Coverity (C/C++, Java, Python, Go...)¹
 - Cppcheck (C/C++)²
 - Pylint (Python)³
 - RuboCop (Ruby)⁴

¹ <https://scan.coverity.com>

² <http://cppcheck.sourceforge.net/>

³ <https://www.pylint.org>

⁴ <https://docs.rubocop.org>

Error: USE_AFTER_FREE (CWE-825):

libblockdev-2.13/src/plugins/lvm-dbus.c:1163: freed_arg: "g_free"
frees "output".

libblockdev-2.13/src/plugins/lvm-dbus.c:1165: pass_freed_arg: Passing freed
pointer "output" as an argument to "g_set_error".

```
# 1163|         g_free (output);  
# 1164|         if (ret == 0) {  
# 1165|->             g_set_error (error, BD_LVM_ERROR, BD_LVM_ERROR_PARSE,  
# 1166|                 "Failed to parse number from output: '%s'",  
# 1167|                 output);
```

Code Style

- Coding conventions – naming, code lay-out, comment style. . .
- Language specific (PEP 8⁵), project specific (Linux kernel coding style⁶) or library/toolkit specific (GTK coding style⁷).
- Automatic checks using specific tools (pycodestyle) or (partially) by the static analysis tools.

⁵ <https://www.python.org/dev/peps/pep-0008/>

⁶ <https://www.kernel.org/doc/html/v5.3/process/coding-style.html>

⁷ <https://developer.gnome.org/programming-guidelines/stable/c-coding-style.html.en>



Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Readability counts.

Tim Peters

The Zen of Python

Linux kernel coding style

<https://www.kernel.org/doc/html/v4.10/process/coding-style.html>

Don't put multiple statements on a single line unless you have something to hide:

```
if (condition) do_this;  
do_something_everytime;
```

The preferred form for allocating an array is the following:

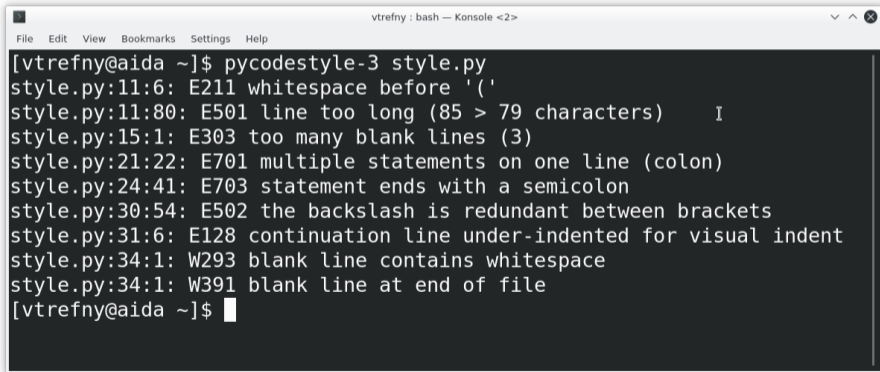
```
p = kcalloc_array(n, sizeof(...), ...);
```

Do not unnecessarily use braces where a single statement will do.

```
if (condition)  
    action();
```

[https:](https://gist.github.com/vojtechtrefny/435737417be003873a7f94aa7d53c4d2)

[//gist.github.com/vojtechtrefny/435737417be003873a7f94aa7d53c4d2](https://gist.github.com/vojtechtrefny/435737417be003873a7f94aa7d53c4d2)

A terminal window titled "vtrefny : bash -- Konsole <2>" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The terminal shows the command "pycodestyle-3 style.py" and its output: "style.py:11:6: E211 whitespace before '('", "style.py:11:80: E501 line too long (85 > 79 characters)", "style.py:15:1: E303 too many blank lines (3)", "style.py:21:22: E701 multiple statements on one line (colon)", "style.py:24:41: E703 statement ends with a semicolon", "style.py:30:54: E502 the backslash is redundant between brackets", "style.py:31:6: E128 continuation line under-indented for visual indent", "style.py:34:1: W293 blank line contains whitespace", and "style.py:34:1: W391 blank line at end of file". The prompt "[vtrefny@aida ~]" is visible at the end of the output.

```
vtrefny : bash -- Konsole <2>
File Edit View Bookmarks Settings Help
[vtrefny@aida ~]$ pycodestyle-3 style.py
style.py:11:6: E211 whitespace before '('
style.py:11:80: E501 line too long (85 > 79 characters)
style.py:15:1: E303 too many blank lines (3)
style.py:21:22: E701 multiple statements on one line (colon)
style.py:24:41: E703 statement ends with a semicolon
style.py:30:54: E502 the backslash is redundant between brackets
style.py:31:6: E128 continuation line under-indented for visual indent
style.py:34:1: W293 blank line contains whitespace
style.py:34:1: W391 blank line at end of file
[vtrefny@aida ~]$
```

Python and PEP8



pep8speaks commented on 18 Feb



Hello @vojtechrefny! Thanks for updating this PR. We checked the lines you've touched for [PEP 8](#) issues, and found:

- In the file `copr_builder/copr_builder.py` :

| [Line 31:54: E261](#) at least two spaces before inline comment

- Documentation might be checked in the same way code is.
- Similar style documents and tools for checking documentations exist (for example PEP 257⁸ and pydocstyle⁹ for Python).
- In some cases wrong or missing documentation (docstrings in the code) can lead to a broken build or missing features.

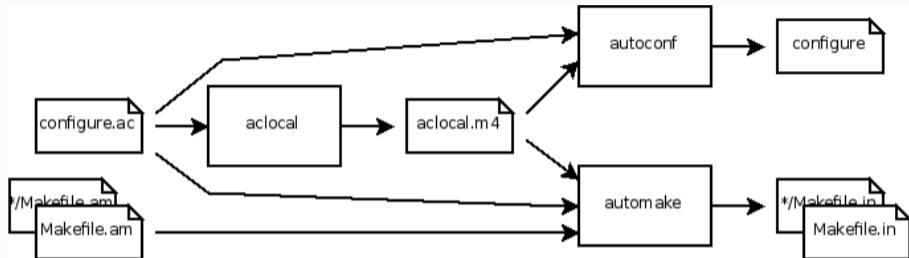
⁸ <https://www.python.org/dev/peps/pep-0257/>

⁹ <http://www.pydocstyle.org>

Build

- Building the project, a preparation to run the test suite.
- Depends on language – mostly no-op for interpreted languages, more complicated for compiled ones.
- Build in the CI environment can detect issues with dependencies.
- Builds on different architectures can help detect issues related to endianness or data types sizes.

- Helps creating portable source packages.
- Two steps:
 - configure (scans the build environment)
 - make (compiles the source)
- Complicated for developers, easy for users.
- Takes care of dependency checking, dynamic linking, installation destinations etc.



I saw a book entitled "Die GNU Autotools" and I thought "My feelings exactly". Turns out the book was in German.

Tim Martin¹⁰

¹⁰<https://twitter.com/timmartin2/status/23365017839599616>

Image source: <https://developer.gnome.org/anjuta-build-tutorial/stable/create-autotools.html.en>

Tests

- Running tests that are part of the project.
- New tests should be part of every change to the codebase.
 - New features require new unit and integration tests.
 - Bug fixes should come with a regression test.
- For some project (like libraries) running test suites of their users might be an option.

- Code coverage (or Test coverage) represents how much of the code is covered by the test suite.
- Usually percentual value that shows how many lines of the code were “visited” by the test.
- Generally a check that all functions and branches are covered by the suite.
- Used as a measure of the test suite “quality”.

Coverage

```
1 def div(a, b):  
2     if b == 0:  
3         raise ValueError  
4     else:  
5         return a / b  
6  
7 assert div(2, 2) == 1
```

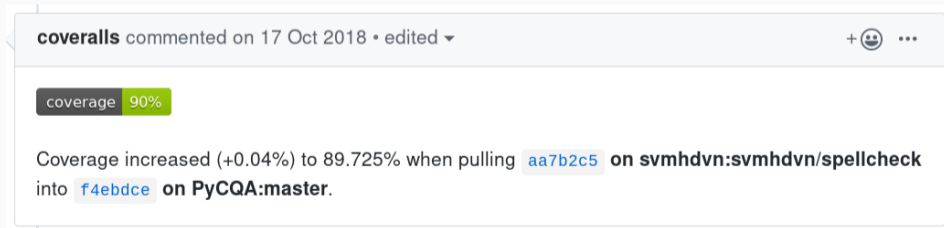
```
$ coverage3 report -m
```

Name	Stmts	Miss	Cover	Missing

div.py	5	1	80%	3

Resulting coverage is 80 %, because 1 of 5 statements is not covered.

- Automated coverage tests might be part of the CI.
- Decrease in coverage can be viewed as a reason to reject contribution to the project.



A screenshot of a GitHub comment from user 'coveralls' dated 17 Oct 2018. The comment contains a coverage report for a pull request. The report shows a coverage of 90% for the current state. The text below the report states: 'Coverage increased (+0.04%) to 89.725% when pulling `aa7b2c5` on `svmhdvn:svmhdvn/spellcheck` into `f4ebdce` on `PyCQA:master`.'

coveralls commented on 17 Oct 2018 • edited ▾ +😊 ⋮

coverage 90%

Coverage increased (+0.04%) to 89.725% when pulling `aa7b2c5` on `svmhdvn:svmhdvn/spellcheck` into `f4ebdce` on `PyCQA:master`.

Delivery and Deployment

Packaging and publishing

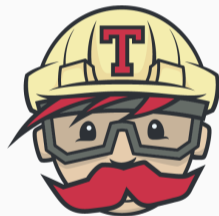
- **Delivery** – releasing new changes quickly and regularly (daily, weekly...).
- **Deployment** – delivery with automated push to production, without human interaction.

- Usually after merging the changes, not for the PRs.
- Building packages, container images, ISO images. . .
- Built packages can be used for further testing (manually by the Quality Assurance or in another CI infrastructure) or directly pushed to production or included in testing/nightly builds of the project.

CI Tools

Demo

- Probably most popular CI service nowadays.
- Can be integrated in your projects on GitHub.
- Free for opensource projects.
- Configured using `.travis.yml` file in the project
- <https://travis-ci.org>





All checks have passed

1 successful check

[Hide all checks](#)



Travis CI - Pull Request Successful in 44s — Build Passed

[Details](#)




This branch has no conflicts with the base branch


Merging can be performed automatically.

Merge pull request



or view [command line instructions](#).

vojtechtrefny / copr-builder  build: passing

[Current](#) [Branches](#) [Build History](#) [Pull Requests](#) More options 


✓ **Pull Request #41** Add a first simple test for copr_builder


Parsing of config files is covered.


[↔ Commit ef796cc](#)


[🔗 #41: Add a first simple test for copr_builder](#)


[📁 Branch master](#)

 **Vojtech Trefny**

 Python

 #25 passed

 Ran for 44 sec

 3 days ago

[🔄 Restart build](#)

- Automation system, not a “true” CI/CD tool.
- Can automatically run given tasks on a node or set of nodes.
- Tasks can be started on time basis or triggered by an external event (like new commit or PR on GitHub).
- <https://jenkins.io/>



- Complex CI system with task to deliver an “Always Ready Operating System”.
- Packages are tested after every change and “gated” if the CI pipeline fails.
- The goal is to prevent breaking the distribution. CI will stop the broken package before it can affect the distribution.



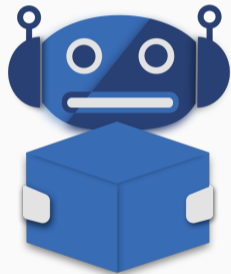


package-tests - 5m 19s



✓ > Currently checking if package tests exist — Print Message	<1s
✓ > Deleting old packages	<1s
✓ > Cloning https://src.fedoraproject.org/rpms/vim/ into the f30 branch	3s
✓ > rpm -q standard-test-roles — Checking if standard-test-roles are installed	<1s
✓ > Getting list of tags	2s
✓ > Print Message	<1s
✓ > Print Message	<1s
✓ > CI Notifier	5s
✓ > Print Message	<1s
✓ > CI Notifier	5s
✓ > Creating directory /workDir/workspace/fedora-f30-build-pipeline/package-tests	<1s
✓ > /tmp/package-test.sh — Shell Script	4m 33s
✓ > logs/ — Verify if file exists in workspace	<1s

- Tool for integrating upstream projects to Fedora.
- RPM packages are automatically build on every pull request.
- New releases can be automatically build and pushed to Fedora.





packit-as-a-service bot commented 24 days ago



Congratulations! One of the builds has completed. 🎉

You can install the built RPMs by following these steps:

- `sudo yum install -y dnf-plugins-core` on RHEL 8
- `sudo dnf install -y dnf-plugins-core` on Fedora
- `dnf copr enable packit/storaged-project-blivet-gui-157`
- And now you can install the packages.

Please note that the RPMs should be used only in a testing environment.

Questions

Thank you for your attention.

<https://github.com/crocs-muni/open-source-development-course>