Open Source Development Course

Lifecycle of open source contribution, project management

Vojtěch Trefný
vtrefny@redhat.com

24. 3. 2021

 twitter.com/vojtechtrefny
 github.com/vojtechtrefny
 gitlab.com/vtrefny

Open source contribution lifecycle

1. Issue

Reporting your own bug or RFE or picking an existing issue to work on.

2. Fork

Getting the code to work in your own "workspace".

3. Development

The easy part :-)

4. Tests

CI test results and/or manual testing.

5. Code review

Code review by the maintainer and making changes.

6. Rebase & merge

Adjust to the current master and merging the changes.

Disclaimer

- These topics are usually different for each project, always check contributor guidelines for specific use cases and workflows.

- This part of the lecture is highly based on GitHub workflow. It applies to similar services like GitLab as well, but projects that don't use these "advanced" code hosting services and rely mostly on emails and mailing list are very different.

Contributing guidelines

Pull Request Checklist

Before sending your pull requests, make sure you followed this list.

- Read contributing guidelines.
- Read Code of Conduct.
- Ensure you have signed the Contributor License Agreement (CLA).
- Check if my changes are consistent with the guidelines.
- Changes are consistent with the Coding Style.
- Run Unit Tests.

`https://github.com/tensorflow/tensorflow/blob/master/CONTRIBUTING.md`

Issue trackers

- Code hosting services like GitHub and GitLab offer integrated issues trackers.
- Some projects use separate tools like Bugzilla or JIRA for tracking bugs and RFEs.

- Issues allow you to discuss the bug/feature with other developers and users.
- Starting with an issue or bug report is not necessary but not doing that increases the risk of your contribution being ignored or rejected.

- GitHub helps first-time contributors to discover *good first issues* to help the project with.

https://github.com/microsoft/vscode/contribute

GitHub Issues
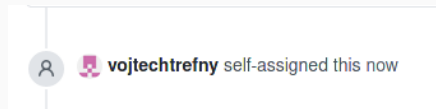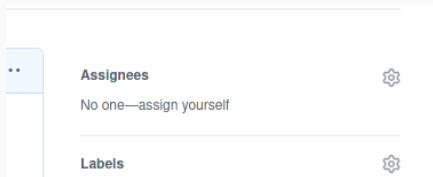
- Do not forget to assign the issue to yourself to let others know you are working on it.

- A fork is your own "copy" of the upstream repository.
- On GitHub (and similar sites) having a fork is needed to be able to open a *pull request* later.
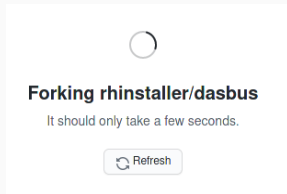
Fork

- It's a good practice to clone the upstream repository and add your fork as a second *remote*.
- This will help you when *rebasing* to the latest upstream version.

```
$ git remote -v
origin    git@github.com:rhinstaller/dasbus.git (fetch)
origin    git@github.com:rhinstaller/dasbus.git (push)
vtrefny_GH    git@github.com:vojtechrefny/dasbus.git (fetch)
vtrefny_GH    git@github.com:vojtechrefny/dasbus.git (push)
```

Pull request

- After committing your changes, simply push them to your fork and open a *pull request* (in GitLab terminology a *merge request*) using the link provided by the *git push hook*.

```
$ git push vtrefny_GH master_my-feature
...
remote:
remote: Create a pull request for 'master_my-feature' on GitHub by visiting:
remote:        https://github.com/vojtechtrefny/dasbus/pull/new/master_my-feature
remote:
To github.com:vojtechtrefny/dasbus.git
 * [new branch]      master_my-feature -> master_my-feature
```

Pull request

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also **compare across forks**.

⇅ base repository: rhinstaller/dasbus ▾ | base: master ▾ | ← | head repository: vojtechtrefny/dasbus ▾ | compare: master_my-shiny-new-feature ▾

✓ **Able to merge.** These branches can be automatically merged.

My shiny new feature

| Write | Preview | | | H B I | ⋮≡ <> 🔗 | ☰ ≔ ☑ | @ ↗ ↩▾ |

Fixes: #50

Attach files by dragging & dropping, selecting or pasting them.

☑ **Allow edits by maintainers** ⓘ

Create pull request ▾

Automated tests

- If the project has CI configured, automated tests will run on your PR.
- Multiple different tests and checks, including static analysis and builds, may be applied.
- Some projects can reject or ignore PRs with failing tests.
- More about tests and CI next week.

All checks have passed
15 successful checks

Some checks were not successful
4 failing and 11 successful checks

Code review

- During code review, other project contributors and maintainers will review your changes to either approve or reject them or request changes.
- When changes are requested you can either change the existing commits or add *fix* commits that will be *squashed* later.
- At least one *approve* review from a project maintainer is usually needed for the change to be merged.
- For more complicated changes, long back and forth reviews with multiple discussions are not uncommon.
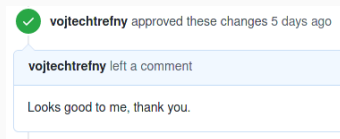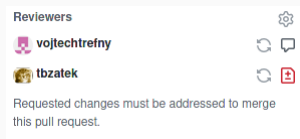
Code review


```
src/udiskslinuxblock.c   Outdated                                    Hide resolved
1780    -        goto out;
        1780 +        {
        1781 +            if (g_error_matches (*error, G_FILE_ERROR, G_FILE_ERROR_NOENT))
        1782 +                contents = g_strdup ("");
```

tbzatek on Jan 8 `Member`

I guess `error` should be cleaned up then?

mvollmer on Jan 8 `Author` `Contributor`

Yes, good point.

mvollmer on Jan 8 `Author` `Contributor`

Done.

tbzatek on Jan 8 `Member`

Thanks!

Reply…

Unresolve conversation   **tbzatek** marked this conversation as resolved.
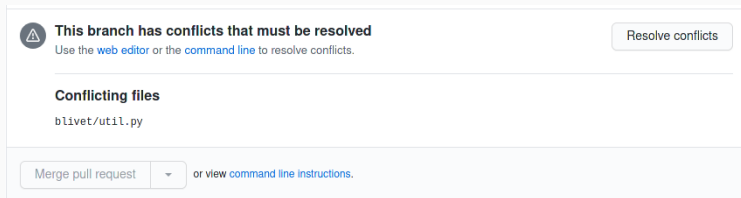
udiskslinuxblock: Survive a missing /etc/crypttab ···   ✕ b79f684

mvollmer force-pushed the `mvollmer:survive-missing-crypttab` branch from **ba57d73** to **b79f684** on Jan 8

✓ **tbzatek** approved these changes on Jan 8   View changes

Rebase & merge

- If the CI and review phases take a long time, merge conflicts can occur, especially in more active projects.
- `git` can solve some conflicts automatically, but you might need to resolve some manually.
- Pull requests with conflicts cannot be merged.

Software dependencies

Software dependencies

- Using existing libraries and tools can save a lot of time when building a new project.

- Managing third party dependencies can be tricky, but when done properly, it's better and safer than writing code from scratch.

- When working with dependencies it's crucial to make sure your
  - code is up to date,
  - system is secure,
  - service/project works as intended.

Direct dependencies

Libraries that your code directly depends upon. These require some effort to control but are sort of manageable.

Transitive dependencies

Dependencies of the dependencies. Usually quite hard to control.

Third party dependencies

A special kind. These are the dependencies that you don't own and that are not part of your organization.
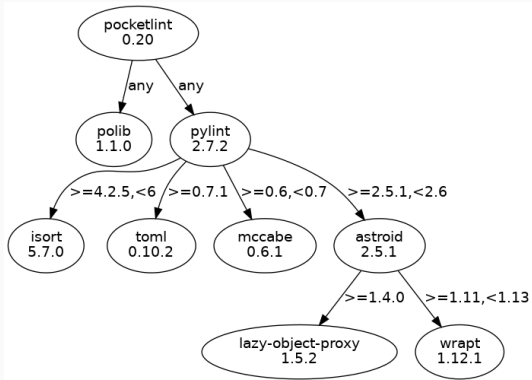
```
pocketlint
  - polib [Any]
  - pylint [Any]
    - astroid [>=2.5.1,<2.6]
      - lazy-object-proxy [>=1.4.0]
      - wrapt [>=1.11,<1.13]
    - isort [>=4.2.5,<6]
    - mccabe [>=0.6,<0.7]
    - toml [>=0.7.1]
```

Dependency management

- Dependencies are usually managed using a project or language-specific tool which takes care of resolving and pulling in third-party dependencies, including transitive dependencies.
- Dependencies can be downloaded during the build (for build and built-in dependencies) or project or service installation or deployment.
- Dependencies can be available either in a public or project-specific repository.
- Some dependency management can also detect required dependencies from the code.

- Different package repositories exist for different programming languages or frameworks:
  - Python Package Index (PyPI) for Python with `pip` tool.[1]
  - Rust crate registry for Rust with `cargo` tool.[2]
  - Node Package Manager (NPM) with `npm` tool.[3]

---

[1] https://pypi.org
[2] https://crates.io
[3] https://www.npmjs.com

pocketlint 0.20

`pip install pocketlint`

✓ Latest version

Released: Aug 30, 2019

```
setup(
    name='pocketlint', version='0.20',
    description='Support for running ...
    url='https://github.com/...
    install_requires=['pylint', 'polib'],
    ...
)
```

```
$ pip install pocketlint
Collecting pocketlint
  Using cached pocketlint-0.20-py3-none-any.whl (32 kB)
Collecting pylint
  Using cached pylint-2.7.2-py3-none-any.whl (342 kB)
Collecting polib
  Using cached polib-1.1.0-py2.py3-none-any.whl (25 kB)
...
Collecting astroid<2.6,>=2.5.1
  Using cached astroid-2.5.1-py3-none-any.whl (222 kB)
Collecting wrap<1.13,>=1.11
  Using cached wrapt-1.12.1.tar.gz (27 kB)
Collecting lazy-object-proxy>=1.4.0
  Using cached lazy_object_proxy-1.5.2-cp39-cp39-manylinux1_x86_64.whl (53 kB)
Installing collected packages: mccabe, isort, toml, wrap, lazy-object-proxy,
        astroid, pylint, polib, pocketlint                      20 / 29
```

What could go wrong?

{* SOFTWARE *}

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

In 2016 Azer Koçulu unpublished more than 250 of his packages from NPM, including a very popular `left-pad` package which many big JavaScript projects (including React) depended on.[4] [5]

---

[4] https://www.theregister.com/2016/03/23/npm_left_pad_chaos/
[5] https://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm

What could go wrong?

Security vulnerabilities (CVE)

Old versions of dependencies with unfixed vulnerabilities.

Version conflicts

Circular dependencies and dependency hell.

Missing/removed dependency

Packages removed from repositories.

API/ABI changes

New versions with backwards-incompatible changes.

Broken third-party dependencies

Bugs in code that is not under your control.

License conflicts

Incompatible open source licenses.

- Two or more packages depending on the same package but a different version.
- Unsolvable if multiple versions of the same package cannot be installed together.
- Sometimes referred to as "dependency hell" (especially in relation to distributions and solving of dependencies during package installation or upgrade).
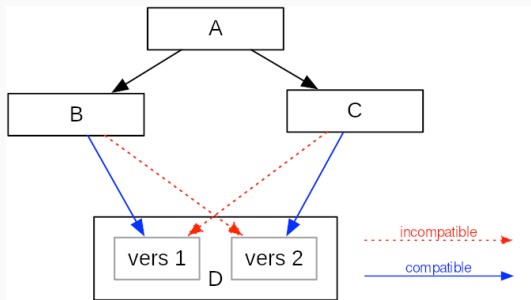
API/ABI stability

Version	Date	Soname	Change Log	Backward Compat.	Added Symbols	Removed Symbols
2.3.4	2020-09-03	12	N/A	100%	0	0
2.3.3	2020-05-28	12	N/A	100%	0	0
2.3.2	2020-04-30	12	N/A	100%	0	0
2.3.1	2020-03-12	12	N/A	100%	0	0
2.3.0	2020-02-02	12	N/A	100%	8 new	0
2.2.2	2019-11-01	12	N/A	100%	0	0
2.2.1	2019-09-06	12	N/A	100%	0	0
2.2.0	2019-08-15	12	N/A	100%	4 new	0
2.1.0	2019-02-08	12	N/A	100%	10 new	0

`https://abi-laboratory.pro/?view=timeline&l=cryptsetup`

- Backward incompatible changes should happen only in major versions.
- Keeping compatibility with multiple API versions require additional code and tests.
- For compiled languages with dynamic libraries, ABI stability is also crucial. Rebuilding the entire project because of a single dependency ABI change might not be possible.

- Both security vulnerabilities and "normal" bugs in dependencies can easily compromise your application.
- Recent bugs like Heartbleed[6] in OpenSSL showed that many essential opensource libraries are understaffed and have many unfixed vulnerabilities.
- Bundled dependencies needs to be closely monitored for CVEs and quickly updated.

---

[6] https://heartbleed.com/

Dead and abandoned projects

- Especially smaller projects can very quickly become unmaintained or abandoned.

- Upstream source can be even completely lost after service or domain expiration.

- Upstream can move to a different fork or cease to exist without replacement.

- For actively maintained projects, older stable versions can be abandoned long before your product EOL.

`https://www.open-fcoe.org/git`

Hmm. We're having trouble finding that site.

We can't connect to the server at www.open-fcoe.org.

If that address is correct, here are three other things you can try:

- Try again later.
- Check your network connection.
- If you are connected but behind a firewall, check that Firefox has permission to access the Web.

Try Again

License compatibility

- Not all free and/or opensource licenses are compatible.
- Some licenses require the result to be released under the "stronger" license.
- Different rules might apply for different types of dependencies – copying code vs. linking vs. calling.
- More about licenses in the *Open source licenses* lecture.

License compatibility

		I want to license my code under:					
		GPLv2 only	**GPLv2 or later**	**GPLv3 or later**	**LGPLv2.1 only**	**LGPLv2.1 or later**	**LGPLv3 or later**
I want to copy code under:	**GPLv2 only**	OK	OK [2]	NO	OK: Combination is under GPLv2 only [7]	OK: Combination is under GPLv2 only [7][2]	NO
	GPLv2 or later	OK [1]	OK	OK	OK: Combination is under GPLv2 or later [7]	OK: Combination is under GPLv2 or later [7]	OK: Combination is under GPLv3 [8]
	GPLv3	NO	OK: Combination is under GPLv3 [3]	OK	OK: Combination is under GPLv3 [7]	OK: Combination is under GPLv3 [7]	OK: Combination is under GPLv3 [8]
	LGPLv2.1 only	OK: Convey copied code under GPLv2 [7]	OK: Convey copied code under GPLv2 or later [7]	OK: Convey copied code under GPLv3 or later [7]	OK	OK [6]	OK: Convey copied code under GPLv3 [7][8]
	LGPLv2.1 or later	OK: Convey copied code under GPLv2 [7][1]	OK: Convey copied code under GPLv2 or later [7]	OK: Convey code under GPLv3 or later [7]	OK [5]	OK	OK
	LGPLv3	NO	OK: Combination is under GPLv3 [8][3]	OK: Combination is under GPLv3 [8]	OK: Combination is under GPLv3 [7][8]	OK: Combination is under LGPLv3 [4]	OK: Combination is under LGPLv3

Questions

Thank you for your attention.

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