Open Source Development Course

Project Lifecycle & Dependency Management

Marek Čermák <cermakm@redhat.com>

Project release lifecycle

Releases indicate the development status and record the development progress

In order for users and contributors to navigate in the project lifecycle and development status and also for the developers to be able to manage the project, versioning and releases are one of the most important things to do when developing a software.

Red Hat

Different people have different needs

"

As a contributor to a project that is new to me, I always struggle to **navigate** in the contribution guidelines.
Should/Can I even contribute? **How do I contribute** — is there a specific procedure? When is the right time?

———

A contributor

"

When **choosing a project** I want to depend on, I usually look at its development status and decide based on the maturity of the project. Concise documentation is a **+**.

———

A developer

"

A project becomes harder to maintain as it grows. The key to success is having a **well-prepared continuous delivery and integration pipelines**. As a benefit, it protects you and your team from an absolute chaos.

———

**A developer & maintainer**

Red Hat

What is a "project"?

The following definition applies to a Python project, but can be easily translated to other languages:

"Projects" are software components that are made available for integration. Projects include Python libraries, frameworks, scripts, plugins, applications, collections of data or other resources, and various combinations thereof. Public Python projects are typically registered on the Python Package Index.

Semantic versioning

The **version scheme** is used both to describe the distribution version provided by a particular distribution archive, as well as to place constraints on the version of dependencies needed in order to build or run the software.

The canonical public version identifiers MUST comply with the following scheme:

[N!]N(.N)*[{a|b|rc}N][.postN][.devN]

major   minor   pre-release   post release   development release
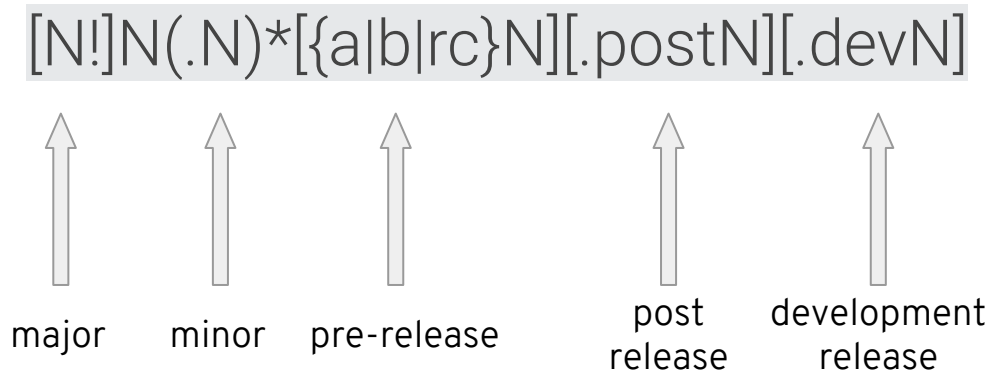
Development status
(Software release lifecycle)

Planning

Ideation phase and defining objectives.

Alpha / Beta

These pre-releases are to support software testing among a limited set of users.

Production/Stable

Stable release used for public consumption. Also called "Final releases".

0.X.Y.devN

0.X.YrcN

X.Y

-

0.X.YaN
0.X.YbN

0.X.Y

Pre-Alpha

These are often used in projects that release "early and often" are not meant for public consumption.

Release Candidate

Candidate for a stable (final) release which is meant for early adopters.

Mature

Usually more than one release or at least one major release.

Planning

Define objectives and the target audience

You gotta know WHAT the project is for and WHO the project is for.

The objectives should we well defined and understandable.

Do the research

Do NOT reinvent the wheel!

If there is an existing project which is similar, see if you can use that one or contribute.

[if not defined] Determine the tools and delivery strategy

These can be publication tools, installation tools and other automation tools used for the development and delivery.

Come up with a POC

Prove the concept. This accounts for a feasibility study as well.

Red Hat
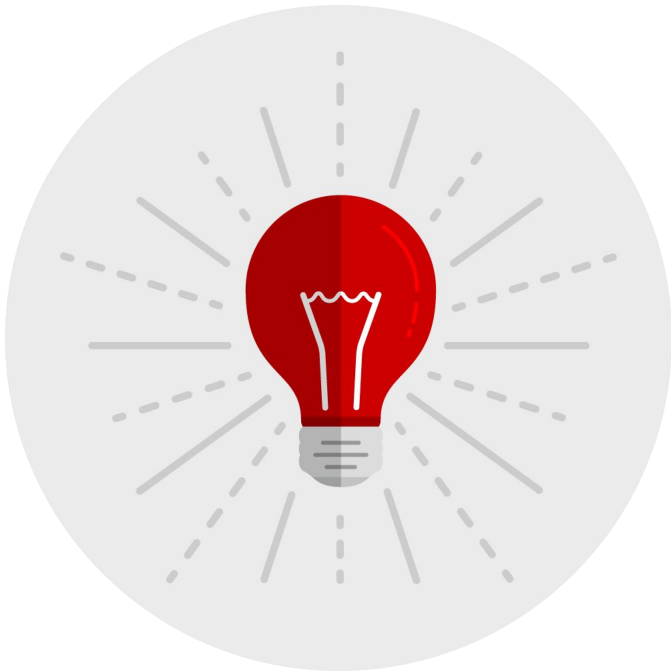
Pre-alpha refers to all activities performed during the software project before formal testing

These activities can include requirements analysis, software design, software development, and unit testing.

Software design

From a concept, through the architecture and implementation details.

Software development

Includes programming, feature implementation, feature enhancements, bug fixes or maintenance (i.e. updates and migrations, etc…)

Unit & Integration testing

Check whether the individual units of source codes function as expected based on a set of determined rules and whether they fit together and function together.

Red Hat

rc

Pre-releases refer to a set of version identifiers which denote a preparation for the final release and are meant for early adopters

Among the pre-releases we include the alpha/beta releases and release candidates

Alpha

The first phase of software testing before releasing it to customers / users. In proprietary software, it is not common for a package in alpha release to be generally available. Alpha usually ends with a feature freeze.

Beta

The software is expected to have bugs which do not directly affect its functionality. The main purpose is to reduce impact on customers / users or to demonstrate and preview a product.
A commercial *betaware* is usually available to limited set of users outside of the organization (**closed beta**) or publicly (**open beta**).

Release Candidates

A beta version with the potential to become the final product ready to be released. Minor fixes to fix certain defects are expected but NO new features or API changes should be made.

Source:
https://en.wikipedia.org/wiki/Software_release_life_cycle

Red Hat

Release (stable, or final release) indicates that the software is stable, tested and ready to be used

Good to go.
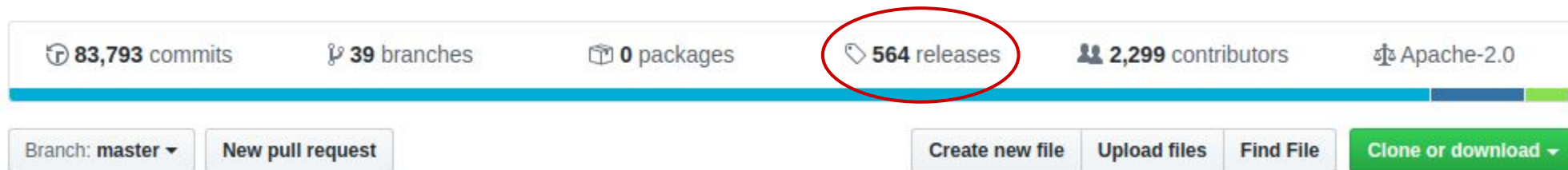
General Availability (GA)

Used mostly for commercial products, but occasionally can be seen in the Open Source world as well. The GA means that the software is available for purchase.

Support

A release should be supported for a certain period of time and in further releases, there should be guarantee of certain backwards compatibility (this is not a rule, but is greatly appreciated).

Source:
https://en.wikipedia.org/wiki/Software_release_life_cycle

Red Hat

So, when to contribute and when to file an issue?

When should I NOT use the project yet?

If a project is well-maintained, it is easy to spot the development status at the first glance.
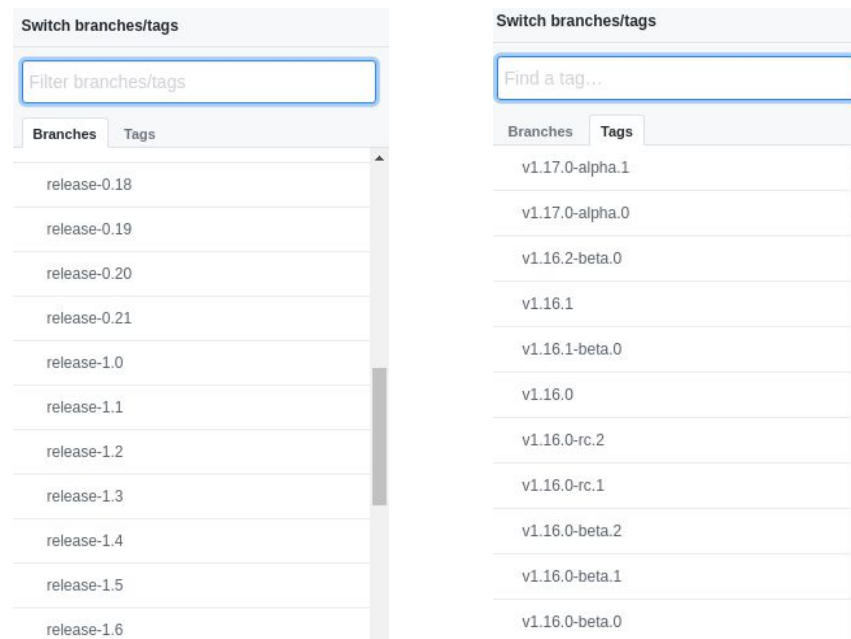
Source:
https://github.com/kubernetes/kubernetes

So, when to contribute and when to file an issue?

When should I NOT use the project yet?

If a project is well-maintained, it is easy to spot the development status at the first glance.

Source:
https://github.com/kubernetes/kubernetes

So, when to contribute and when to file an issue?

When should I NOT use the project yet?

If a project is well-maintained, it is easy to spot the development status at the first glance.

```
classifiers=[
        "Development Status :: 2 - Pre-Alpha",
        "Framework :: IPython",
        "Framework :: Jupyter",
        "License :: OSI Approved :: MIT License",
        "Natural Language :: English",
        "Operating System :: OS Independent",
        "Programming Language :: JavaScript",
        "Programming Language :: Python :: 3.6",
        "Programming Language :: Python :: 3.7",
        "Topic :: Utilities",
],
```

Contribution Guidelines

Guidelines communicate how people should contribute to you project.

Contribution guidelines are a set of recommended practices, or sometimes even required ones, established by a maintainer for the contributors to be followed.

Red Hat

Before contributing to an open source project, make sure to check its contribution guidelines!

Usually, they can be found in a file called CONTRIBUTION.md[0] or, in case of GitHub, they might be integrated to PRs and Issues directly[1]

Verification for both contributors and developers

For both contributors and developers, the guidelines help them verify that they're submitting well-formed pull requests and opening useful issues.

Getting started for contributors

Contributors might struggle to navigate in the project or they may not know where to start contributing, what should the PR or issue look like.

Prevent confusion and save time

For both owners and contributors, contribution guidelines save time and hassle caused by improperly created pull requests or issues that have to be rejected and re-submitted.

Red Hat

Source:
https://help.github.com/assets/images/help/pull_requests/contributing-guidelines.png

As for the WHAT to contribute …
It doesn't have to be code

""

Most people don't know that I actually don't do any real work on the CocoaPods tool itself. My time on the project is mostly spent doing things like documentation and working on branding.

———

@orta

""

I first reached out to the Python development team (aka python-dev) when I emailed the mailing list on June 17, 2002 about accepting my patch. I quickly caught the open source bug, and decided to start curating email digests for the group. They gave me a great excuse to ask for clarifications about a topic, but more critically I was able to notice when someone pointed out something that needed fixing.

———

@brettcannon

Source:
https://opensource.guide/how-to-contribute/

Red Hat

"

Okay, I read the contribution guidelines.

What now?

How do I proceed?

What are the best practices?

—

Ready-to-contribute pessimist Jerry

It all starts with a FORK ...

FORK the repository

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

`git clone`

Clone the repository to your local machine.

Create a branch for the specific purpose

`git checkout -b fix-readme-typo`

Commit your changes

`git commit -a --sign`

Iterate on the issue

Not all fixes are so simple that they can fit into one commit.

`git push`

Push to the remote and you're ready to create a PR against the upstream.

Pull Request

Can you spot an ISSUE?

FORK the repository

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Create a branch for the specific purpose

`git checkout -b fix-readme-typo`

Iterate on the issue

Not all fixes are so simple that they can fit into one commit.

Pull Request

`git clone`

Clone the repository to your local machine.

Commit your changes

`git commit -a --sign`

`git push`

Push to the remote and you're ready to create a PR against the upstream.

Can you spot an ISSUE?

Potentially **lots of** commits!

FORK the repository

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Create a branch for the specific purpose

`git checkout -b fix-readme-typo`

Iterate on the issue

Not all fixes are so simple that they can fit into one commit.

Pull Request

`git clone`

Clone the repository to your local machine.

Commit your changes

`git commit -a --sign`

`git push`

Push to the remote and you're ready to create a PR against the upstream.

Let's squash it!

FORK the repository

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

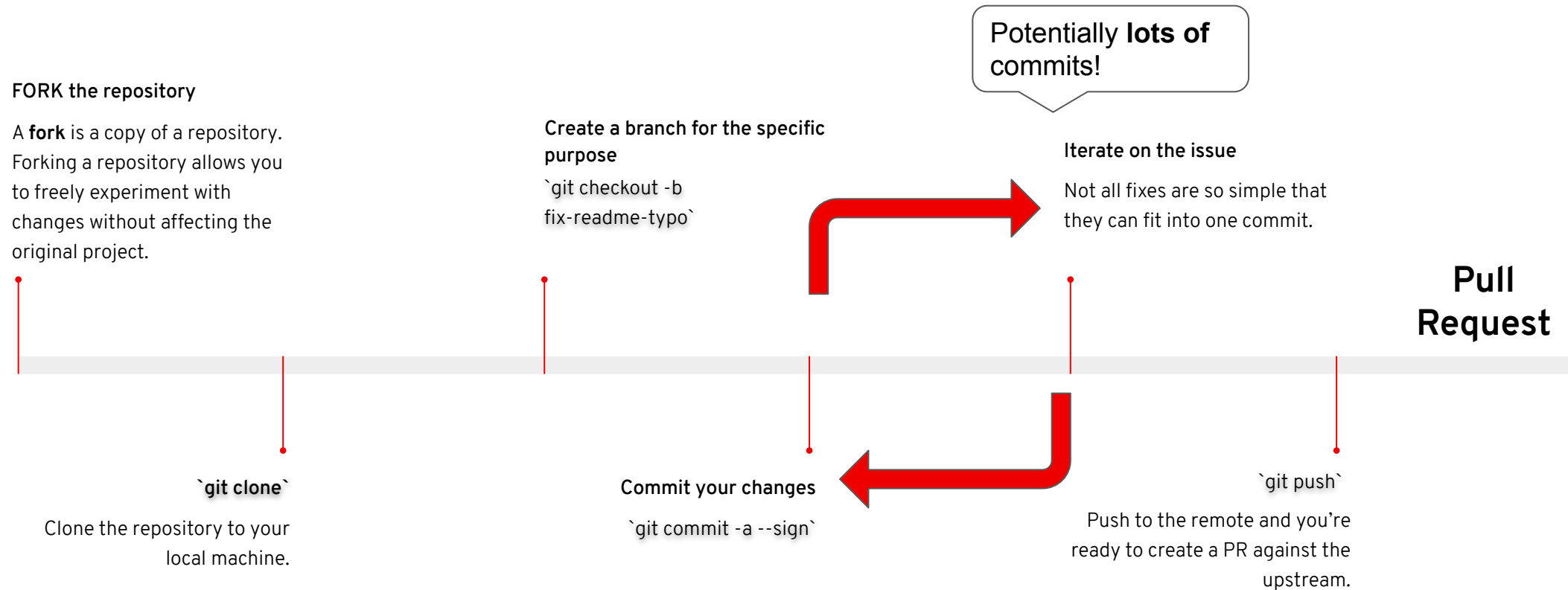**Create a branch for the specific purpose**

`git checkout -b fix-readme-typo`

Iterate on the issue

Not all fixes are so simple that they can fit into one commit.

Pull Request

`git clone`

Clone the repository to your local machine.

Commit your changes

`git commit -a --sign`

`git rebase -i <initial commit>`
`git push`

Squash unnecessary commits, like minor fixes and typos and push them to the remote.

Red Hat

Can you spot an ISSUE?

FORK the repository

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Create a branch for the specific purpose

`git checkout -b fix-readme-typo`

Iterate on the issue

Not all fixes are so simple that they can fit into one commit.

Pull Request

`git clone`

Clone the repository to your local machine.

Commit your changes

`git commit -a --sign`

`git rebase -i <initial commit>` `git push`

Squash unnecessary commits, like minor fixes and typos and push them to the remote.

Red Hat

Pull, squash and push … sounds weird, but does wonders!

FORK the repository

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.
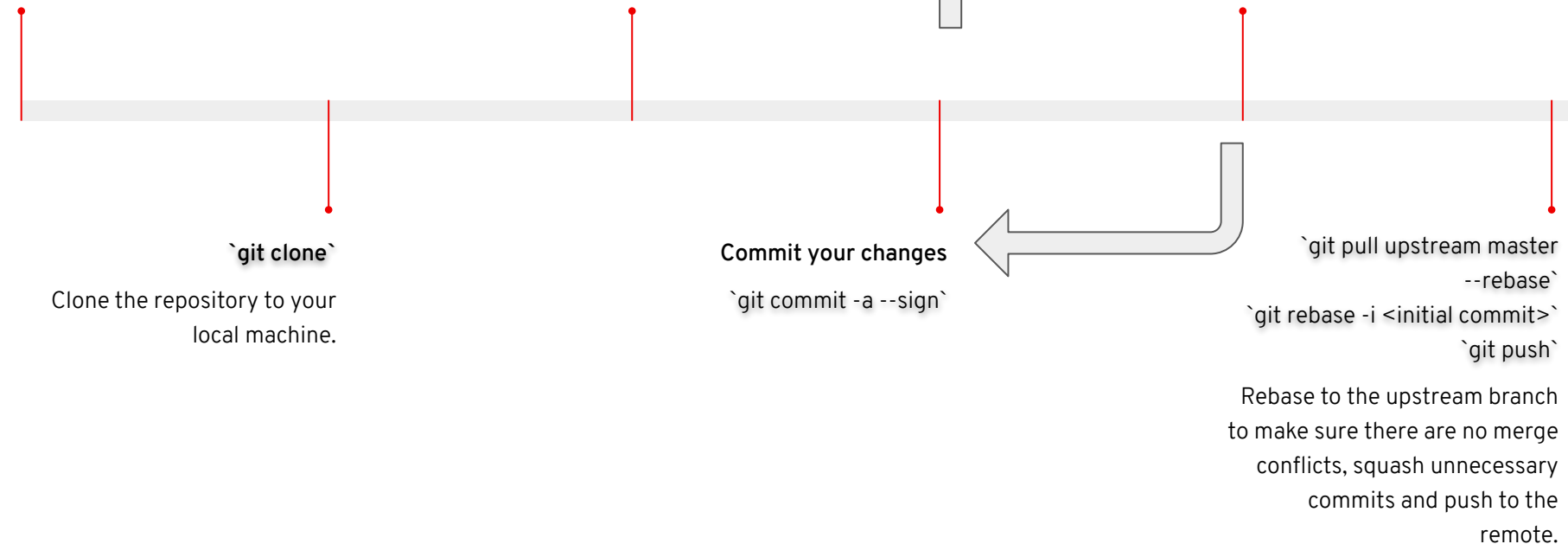
Create a branch for the specific purpose

`git checkout -b fix-readme-typo`

Iterate on the issue

Not all fixes are so simple that they can fit into one commit.

`git clone`

Clone the repository to your local machine.

Commit your changes

`git commit -a --sign`

`git pull upstream master --rebase`
`git rebase -i <initial commit>`
`git push`

Rebase to the upstream branch to make sure there are no merge conflicts, squash unnecessary commits and push to the remote.

Pull Request

There are other useful practices to follow when contributing to the upstream

Document WHAT and WHY

Documenting why the changes have been made and what lead to the decisions you'd made will save the reviewer's time and will increase the chances of your PR being merged. Use provided doc generators, if possible.

Follow the code style

When contributing code, it is good practice to adapt to the project code style (especially for languages with fluid code styles, like JS). Use provided formatters, if possible.

Run tests before submitting the PR and write new ones when introducing new features.

Be extremely careful if changing project dependencies (see further)

Red Hat

Software dependencies

Dependencies are the hell for maintainers, a blessing for developers and the heaven for attackers.

A *dependency* is additional code that you want to call from your program. Adding a dependency avoids repeating work already done: designing, writing, testing, debugging, and maintaining a specific unit of code

Red Hat

Kinds of dependencies

Direct dependencies

Libraries that your code depends upon. These require some effort to control but comparing to the others they 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. Especially hard to control.

Transitive dependencies
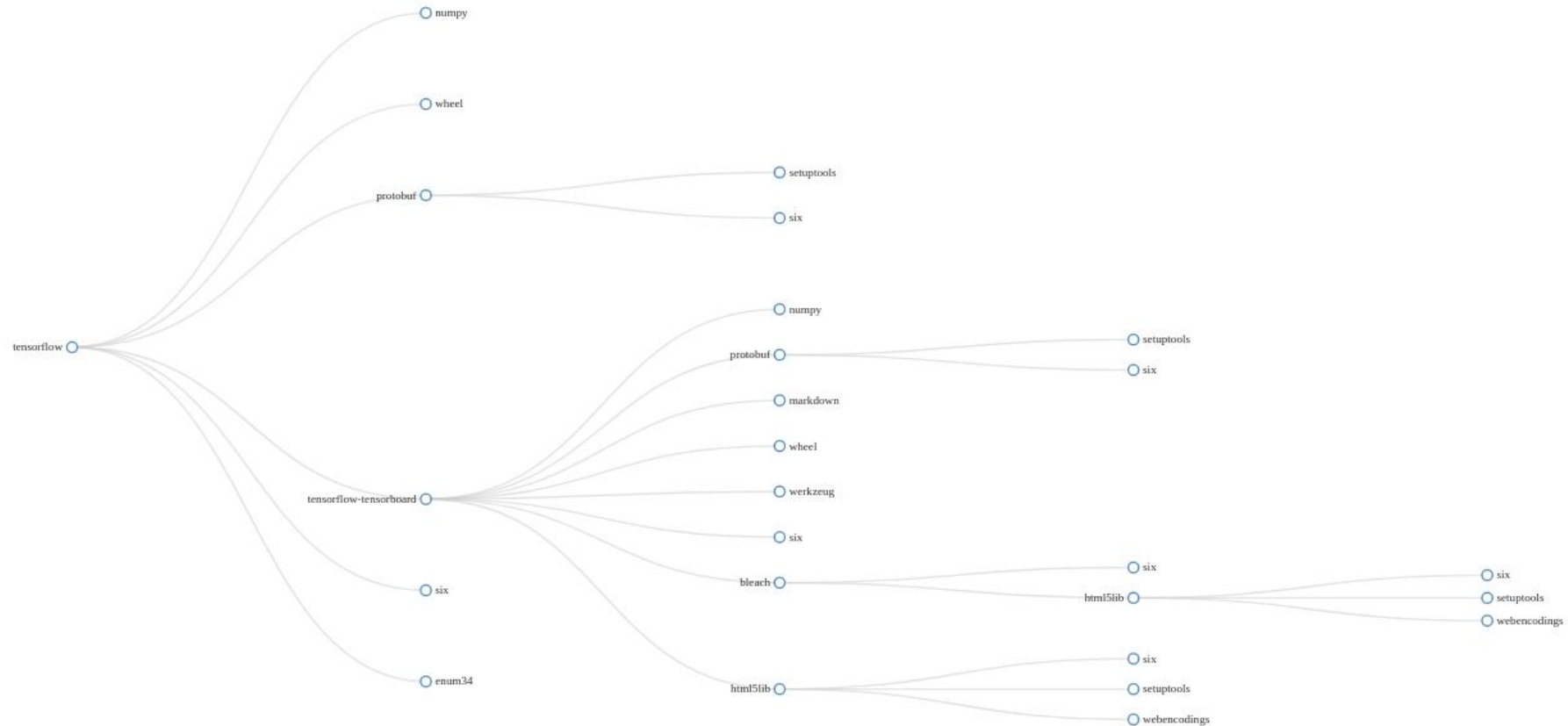
What could go wrong?

A package is code you **download** from the internet. Adding a package as a **dependency** outsources the work of developing that code–designing, writing, testing, debugging, and maintaining–to someone else on the internet, someone you often don't know. By using that code, **you are exposing your own program to all the failures and flaws in the dependency**.

Your program's execution now **literally** *depends* **on code downloaded from this stranger** on the internet.

What could go wrong? You name it …

Security vulnerability (CVE)

Version conflict

Missing/Removed dependency

API Changes

License conflict

Broken third-party dependency

Red Hat

It sounds unsafe …

And it *is*… but it is also necessary to keep the wheel of Open Source spinning!

A note about the security vulnerabilities

What is a "Vulnerability?"

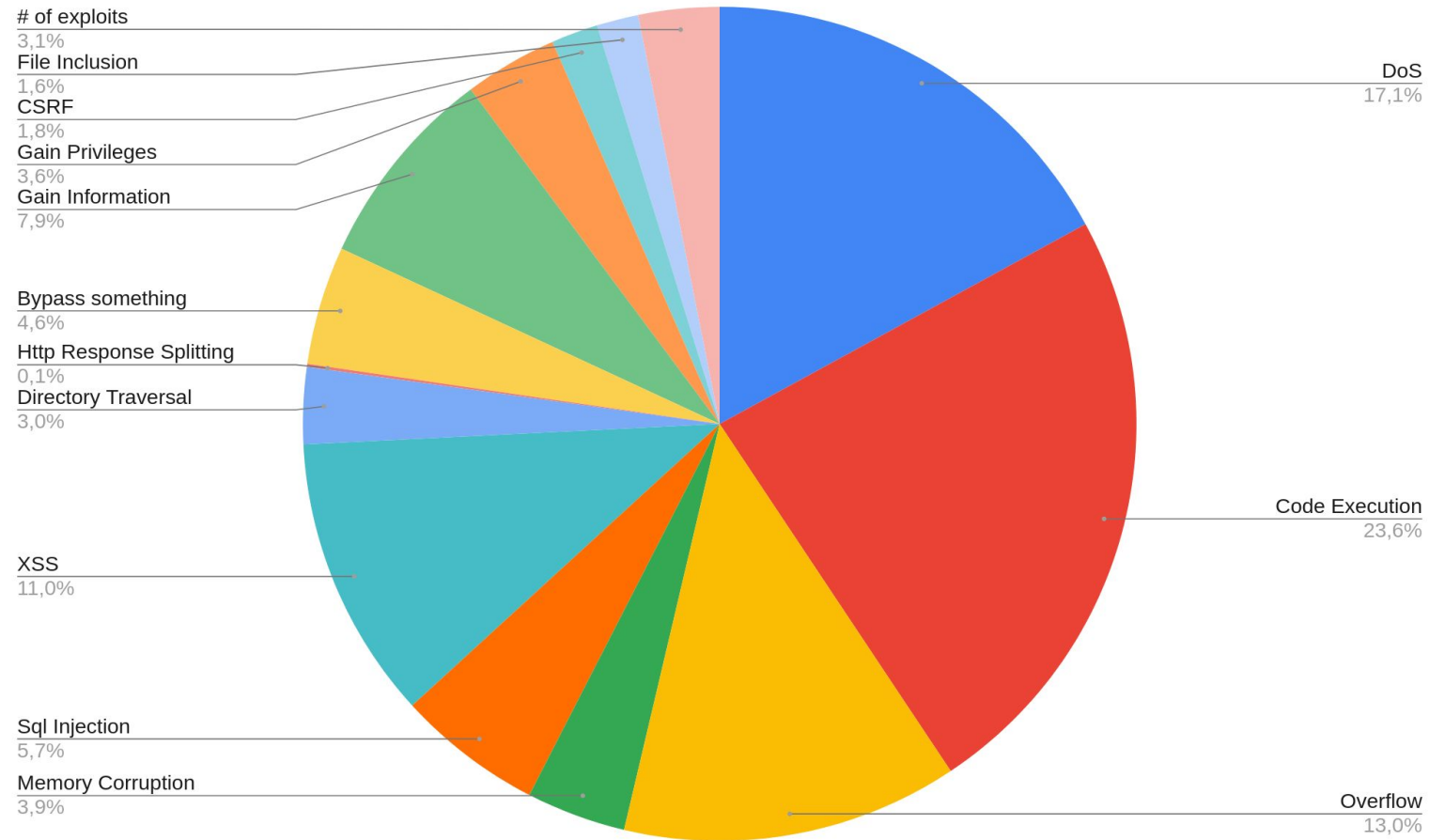An information security "vulnerability" is a mistake in software that can be directly used by a hacker to gain access to a system or network.

What is an "Exposure?"

An information security exposure is a mistake in software that allows access to information or capabilities that can be used by a hacker as a stepping-stone into a system or network.

What is CVE?

CVE is a list of information security vulnerabilities and exposures that aims to provide common names for publicly known problems. The goal of CVE is to make it easier to share data across separate vulnerability capabilities (tools, repositories, and services) with this "common enumeration." Please visit http://cve.mitre.org/about/faqs.html for more information

Common vulnerabilities according to the NVD

# of exploits
3,1%

File Inclusion
1,6%

CSRF
1,8%

Gain Privileges
3,6%

Gain Information
7,9%

Bypass something
4,6%

Http Response Splitting
0,1%

Directory Traversal
3,0%

XSS
11,0%

Sql Injection
5,7%

Memory Corruption
3,9%

DoS
17,1%

Code Execution
23,6%

Overflow
13,0%

34

Source:
https://www.cvedetails.com/vulnerabilities-by-types.php
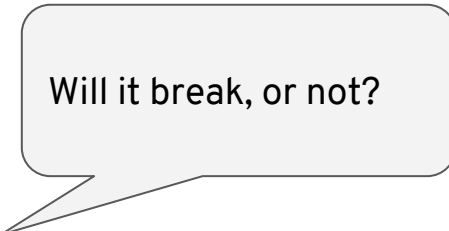
Beware the "dependency hell"

Especially when working with complex systems which have a lot of dependencies, it might be *incredibly* **difficult** to find the "right" **combination of versions** which are actually compatible together.

Sometimes, we might actually reach sort of a "**deadlock**" state if one dependency requires a version of another which is in fact not compatible with the rest of the project, i.e.:

Will it break, or not?

A requires D_a && D_a requires X == 1.13

A requires D_b && D_b requires X == 1.13.5

Red Hat

It might become tedious ...
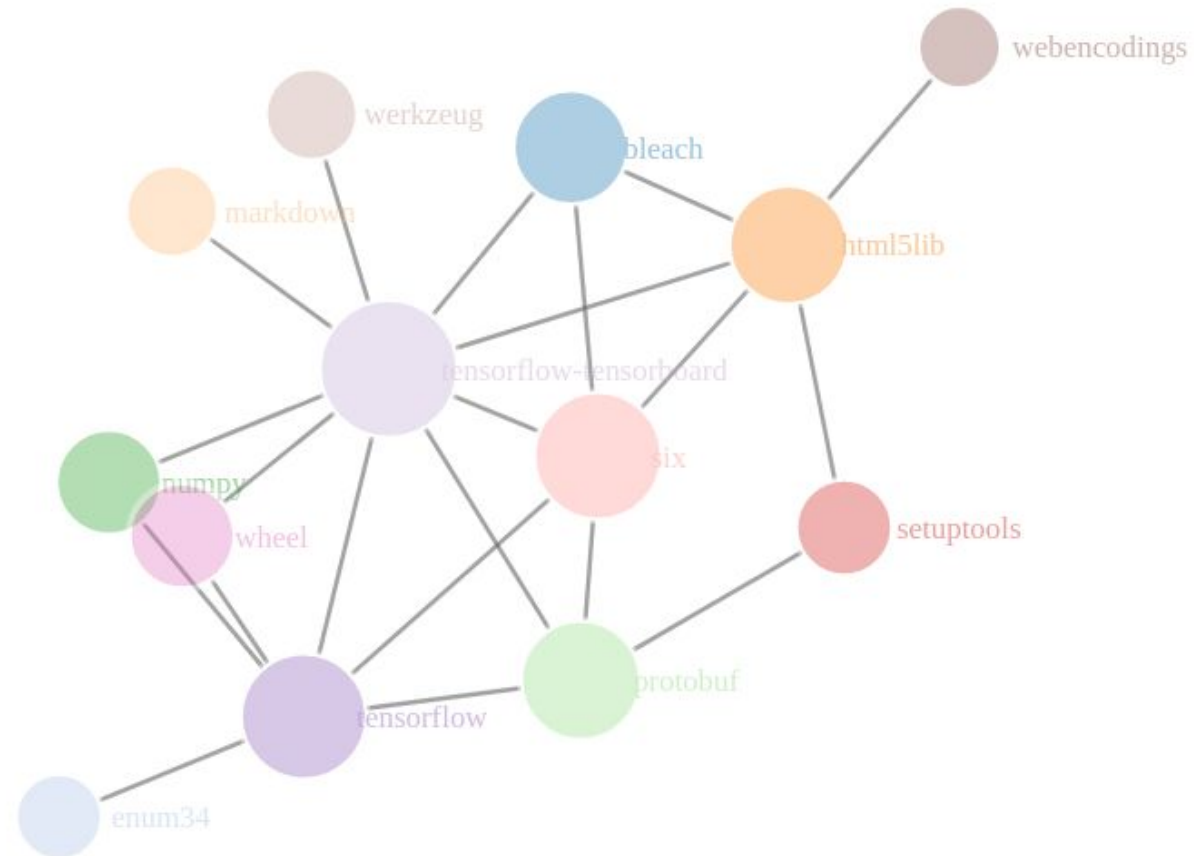
Can you guess the issue?

HINT: centrality, weights

Source:
https://cermakm.shinyapps.io/Tensorflow_Transitive_Dependencies/
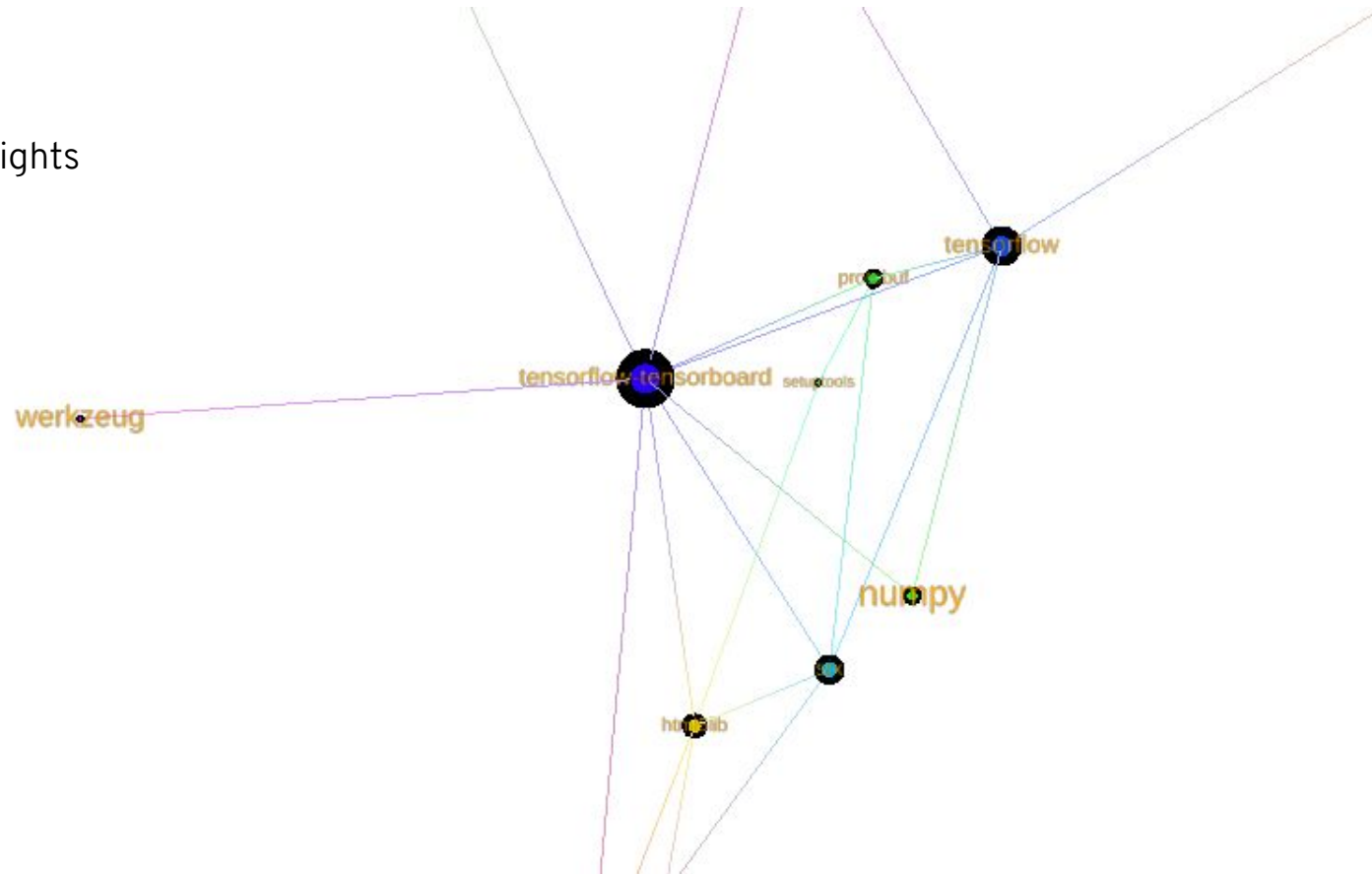
Can you guess the issue?

A dependency can actually have a greater **centrality** and thus be "more important"!

Source:
https://cermakm.shinyapps.io/Tensorflow_Transitive_Dependencies/

So... what can we do?

Red Hat
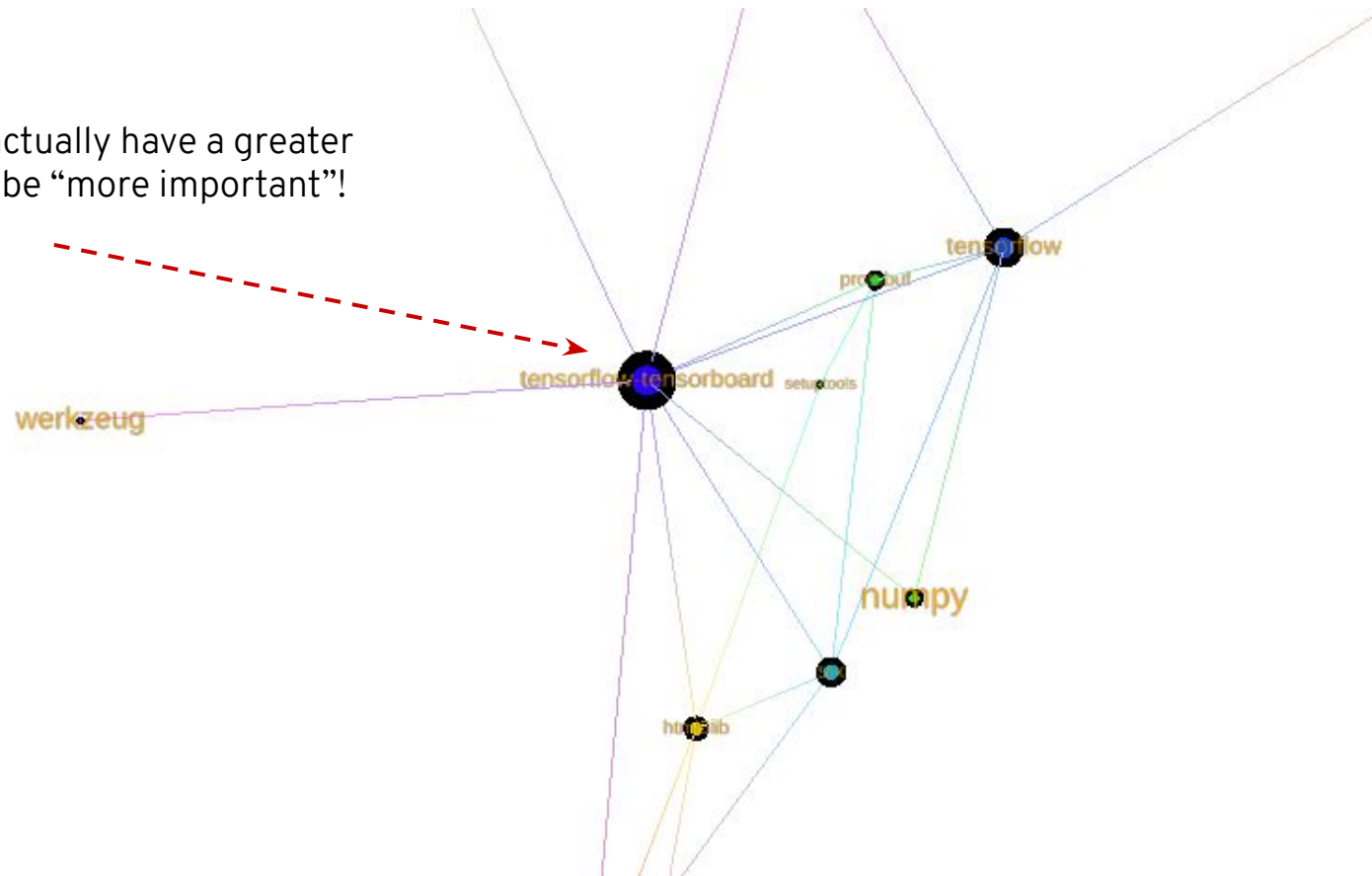
Good practices when managing dependencies

Consider the value of adding the dependency

If introducing the dependency means a few lines of code that you're spared of, do NOT introduce the dependency at all. It is not worth it.

Consider the impact of the dependency

Consider how important the dependency is to your application and treat the dependency accordingly.

Unit TESTS & integration TESTS!

Write unit tests and integration tests especially for functions using a code that you don't own!

Choose a compatible and secure version

Take the time and investigate. Choose a version which is CVE free and is compatible with the rest of the application.

Keep your dependencies up to date

Update the dependencies and keep the code you own up to date with them. Do not rely on the pinned down version.

Regularly watch for CVEs and consult the NVD

Do NOT expose your application. GitHub and specialized software exist to inform you about potential security risks of your application.

Red Hat

And don't ever forget …

TO MAKE SURE THAT THE **LICENSES** ARE COMPATIBLE!

The compatibility is sometimes tricky …

		I want to release a project 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: Convey project under GPLv2 only [7]	OK: Convey project under GPLv2 only [7][2]	NO
	GPLv2 or later	OK [1]	OK	OK	OK: Convey project under GPLv2 or later [7]	OK: Convey project under GPLv2 or later [7]	OK: Convey project under GPLv3 [8]
	GPLv3	NO	OK: Convey project under GPLv3 [3]	OK	OK: Convey project under GPLv3 [7]	OK: Convey project under GPLv3 [7]	OK: Convey project under GPLv3 [8]
	LGPLv2.1 only	OK: Convey code under GPLv2 [7]	OK: Convey code under GPLv2 or later [7]	OK: Convey code under GPLv3 [7]	OK	OK [6]	OK: Convey code under GPLv3 [7][8]
	LGPLv2.1 or later	OK: Convey code under GPLv2 [7][1]	OK: Convey code under GPLv2 or later [7]	OK: Convey code under GPLv3 [7]	OK [5]	OK	OK
	LGPLv3	NO	OK: Convey project and code under GPLv3 [8][3]	OK: Convey code under GPLv3 [8]	OK: Convey project and code under GPLv3 [7][8]	OK: Convey project under LGPLv3 [4]	OK
I want to use a library under:	GPLv2 only	OK	OK [2]	NO	OK: Convey project under GPLv2 only [7]	OK: Convey project under GPLv2 only [7][2]	NO
	GPLv2 or later	OK [1]	OK	OK	OK: Convey project under GPLv2 or later [7]	OK: Convey project under GPLv2 or later [7]	OK: Convey project under GPLv3 [8]
	GPLv3	NO	OK: Convey project under GPLv3 [3]	OK	OK: Convey project under GPLv3 [7]	OK: Convey project under GPLv3 [7]	OK: Convey project under GPLv3 [8]
	LGPLv2.1 only	OK	OK	OK	OK	OK	OK
	LGPLv2.1 or later	OK	OK	OK	OK	OK	OK
	LGPLv3	NO	OK: Convey project under GPLv3 [9]	OK	OK	OK	OK

Source:
https://www.slideshare.net/SamsungOSG/guide-to-open-source-compliance

Red Hat

Continuous {

Integration,

Delivery,

Deployment

}

Introduction to CI/CD

CI/CD are the acronyms that are often mentioned when people talk about modern development practices.[0]

Source:

[0] https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Red Hat

CI/CD is a set of practices which have a significant impact to the way new releases are delivered and maintained.

These are the three main practices to be familiar with.
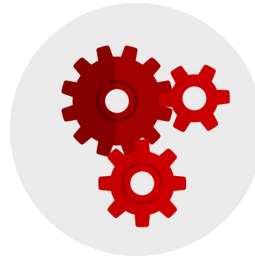
Continuous Integration

Change validation by creating a build and running automated tests against the build. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.

Continuous Delivery

An extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process

Continuous Deployment

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production

Open Source Development Course 2019

Q&A

Red Hat

THANK YOU

 https://github.com/CermakM

 https://www.linkedin.com/in/ai-mcermak/

 https://twitter.com/Marc_Cermak

 https://www.facebook.com/macermak

Open Source Development Course

CI/CD Pipelines

Presenter's Name

Vojtěch Trefný <vtrefny@redhat.com>

Red Hat

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.
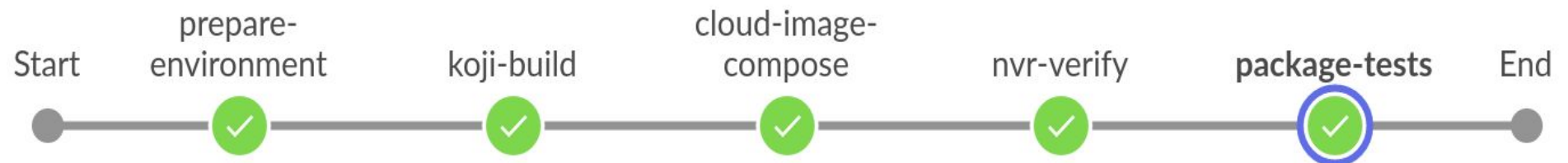
CI/CD Pipeline

Testing environment

Preparation of the environment to run the tests -- deploying a container, starting a VM...

Codestyle

Checking for violations of the language or project style guides.

Tests

Running project test suite or test suites.

Static Analysis

Finding defects by analyzing the code without running it.

Build

Building the project from source.

Packaging and Deployment

Building source archives, packages or container images.

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.

Red Hat

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]

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 and style guides

-   Coding conventions -- naming, code lay-out, comment style...

-   Language specific (PEP 8[1]), project specific (Linux kernel coding style[2]) or library/toolkit specific (GTK coding style[3]).

-   Automatic checks using specific tools (pycodestyle) or (partially) by the static analysis tools.

Source:
[1] https://www.python.org/dev/peps/pep-0008/
[2] https://www.kernel.org/doc/html/v5.3/process/coding-style.html
[3] 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

Red Hat

Linux kernel coding style

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 = kmalloc_array(n, sizeof(...), ...);
```

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

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

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

Python and PEP8

https://gist.github.com/vojtechtrefny/435737417be003873a7f94aa7d53c4d2

Python and PEP8

Documentation style

- Documentation might be checked in the same way code is.

- Similar style documents and tools for checking

  documentations exist (for example PEP 257[1] and

  pydocstyle[2] for Python).

- In some cases wrong or missing documentation (docstrings

  in the code) can lead to a broken build or missing features.

Source:
[1] https://www.python.org/dev/peps/pep-0257/
[2] http://www.pydocstyle.org

Red Hat

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.

GNU Autotools

- 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.

Red Hat
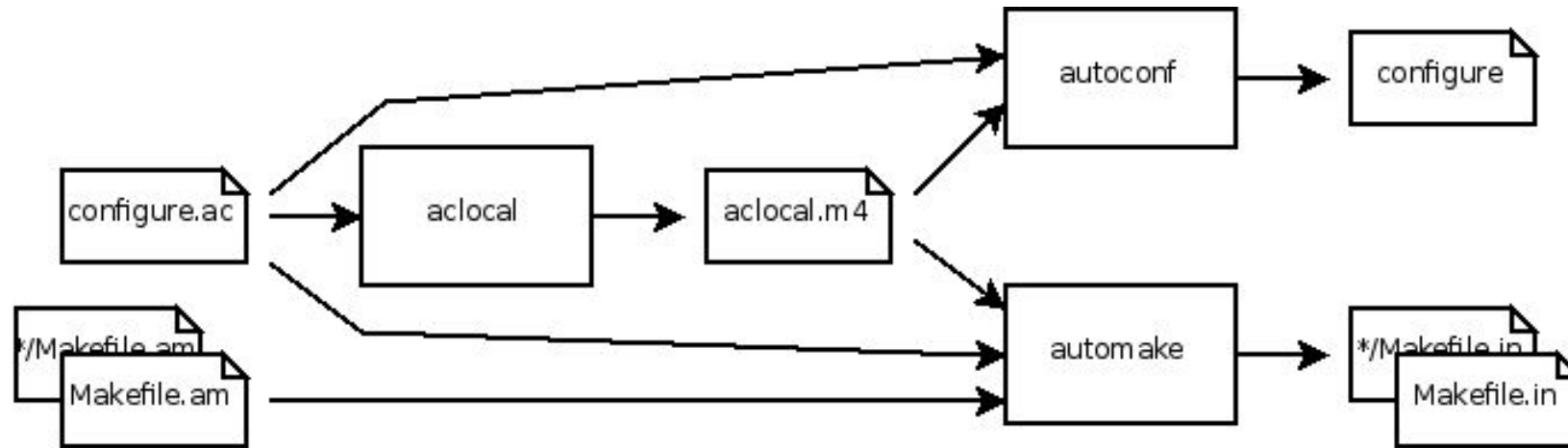
GNU Autotools

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

—

Tim Martin[1]

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.

Red Hat

Coverage

- 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
```

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

$ coverage3 report -m

```
Name      Stmts  Miss Cover  Missing
----------------------------------------
div.py      5      1    80%     3
```

Red Hat

Coverage

- Automated coverage tests might be part of the CI.

- Decrease in coverage can be viewed as a reason to

  reject contribution to the project.

Packaging and publishing

- 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 projects.

Fedora CI

After a change to a package is made, CI pipeline specific
for this package is started.
Both package-specific and generic tests are part of the CI.

Compose

Nightly composes are created from all packages.
Compose can fail for multiple reasons like broken
dependencies or bugs not covered by the test suite.

Installation images and repositories

Daily builds are available in form of installation ISO files.
These are separately tested using openQA automated
test suite.

Red Hat

CI Tools

Demo

Red Hat
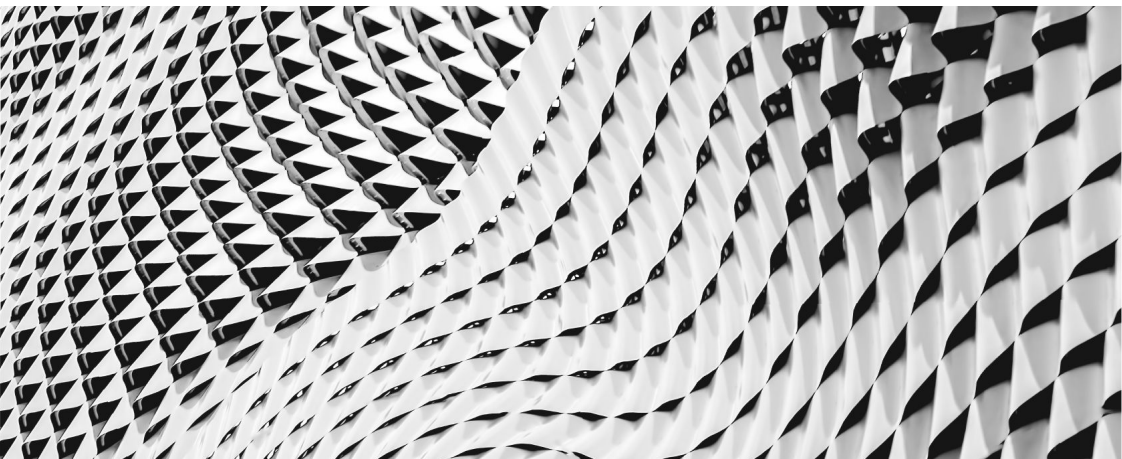
Travis CI

- 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

Red Hat

Travis CI

Travis CI

Jenkins

- 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/

Fedora CI

- 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.

Fedora CI

package-tests - 5m 19s

✓	› Currently checking if package tests exist — Print Message	<1s
✓	› Deleting old packages	<1s
✓	› Cloninghttps://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

Q&A

Red Hat

Open Source Development Course 2019

78

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

in linkedin.com/company/red-hat

f facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat