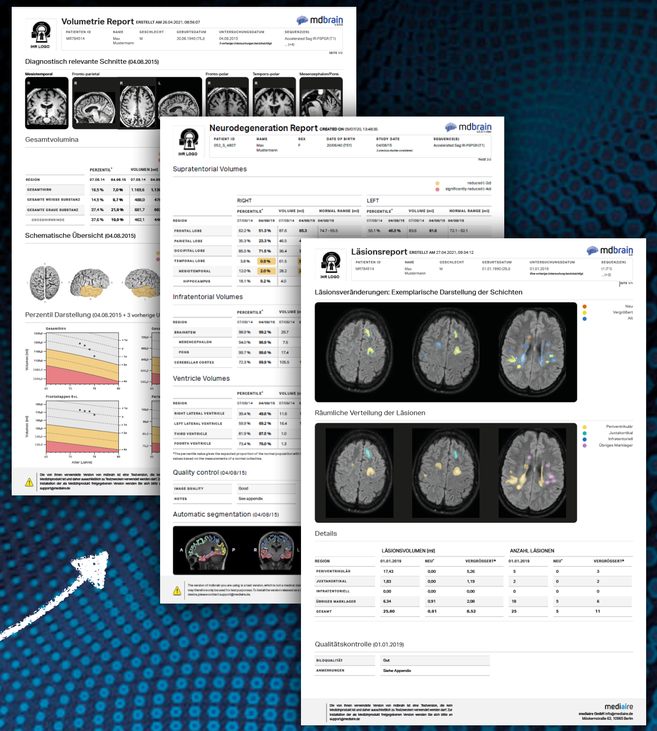
Quitting pip:

How we use git submodules to manage internal dependencies that require fast iteration

Philipp Stephan, mediaire, PyCon 2022

mdbrain

AI-POWERED TOOL FOR
QUANTITATIVE NEURO-MRI



mediaire | DIGITAL INNOVATION IN RADIOLOGY

component_a

component_b

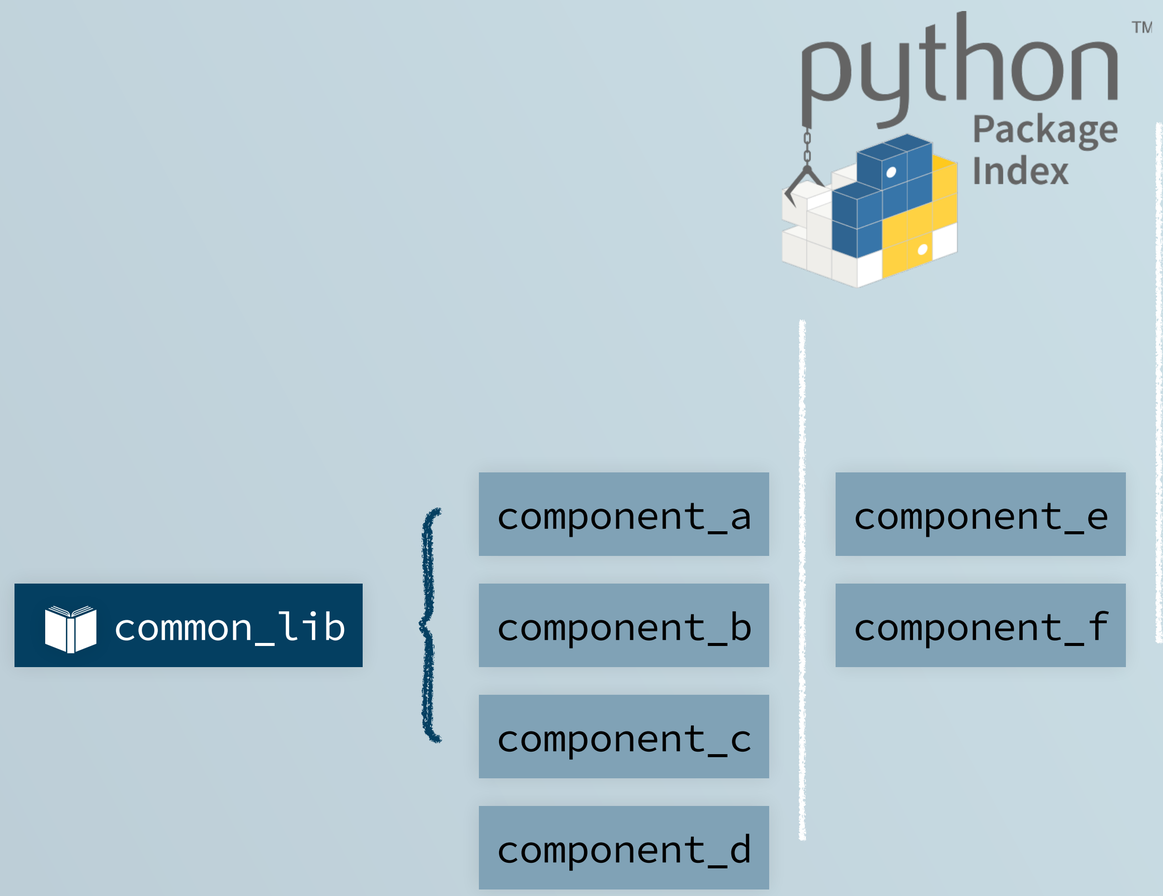
component_c

component_d

numpy

sqlalchemy

matplotlib


python™
Package
Index
common_lib
component_a
component_b
component_c
component_d
component_e
component_f

The Zen of Python

```
>>> import this
```

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.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea—let's do more of those!

There should be one—and preferably only one—obvious way to do it.

package managers

~~easy_install~~
pip

Conda

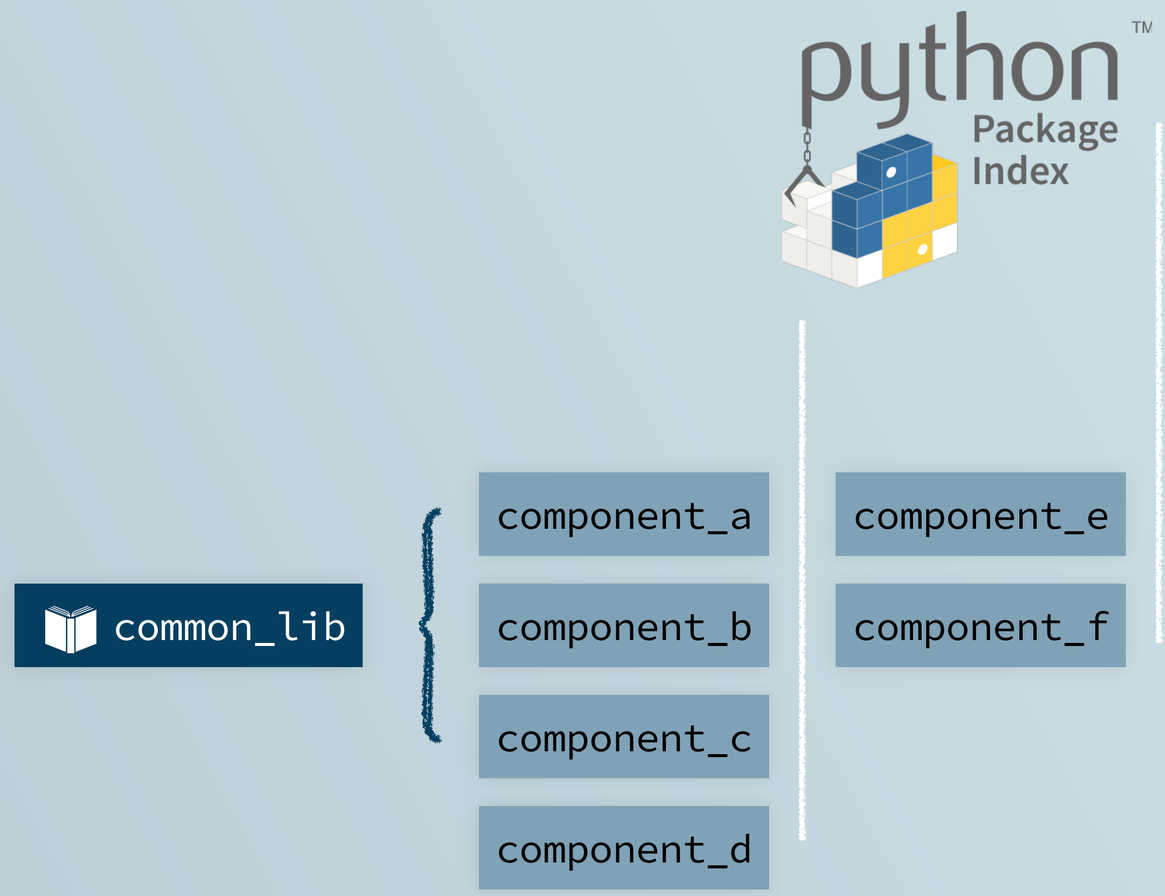
Poetry

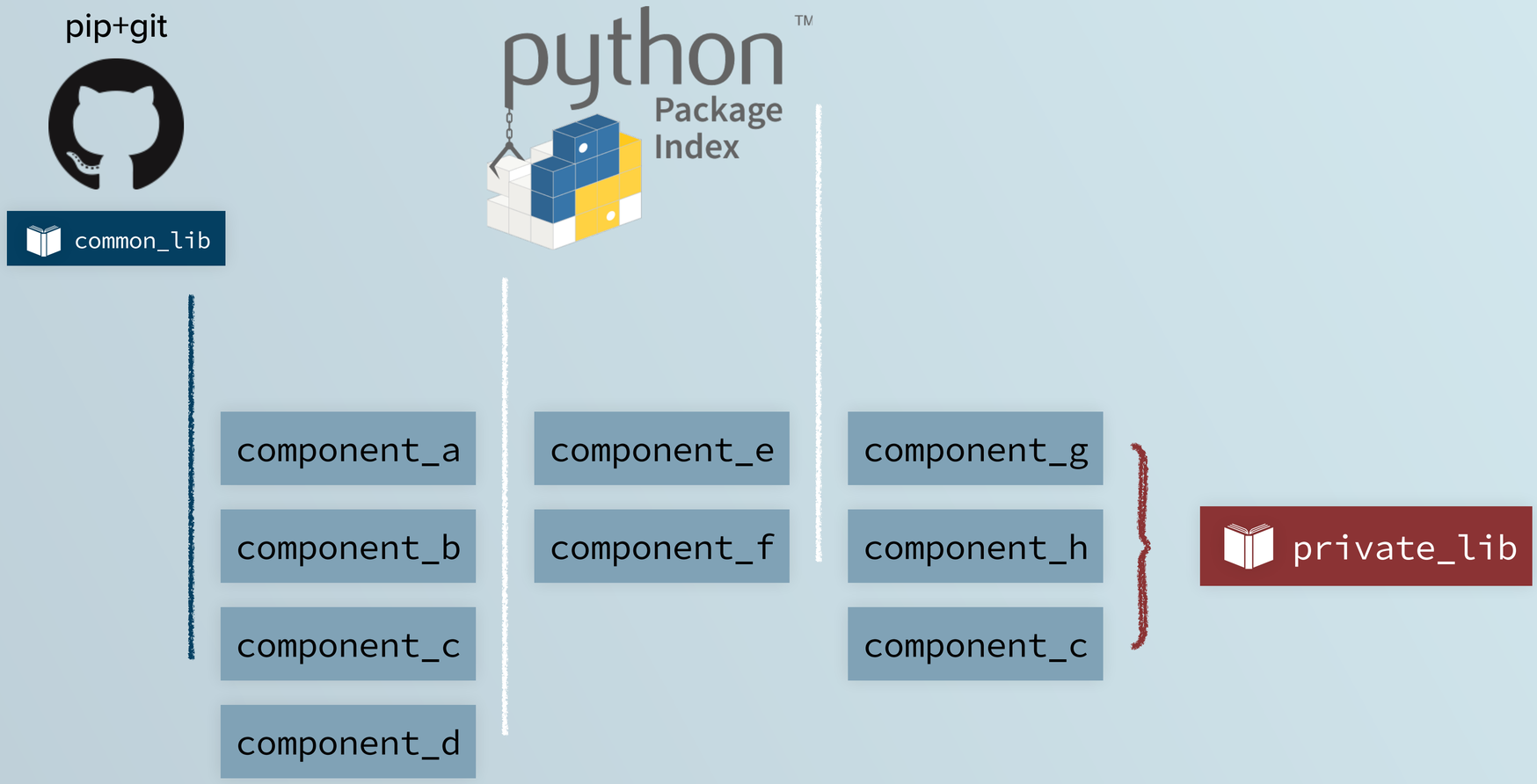
package formats

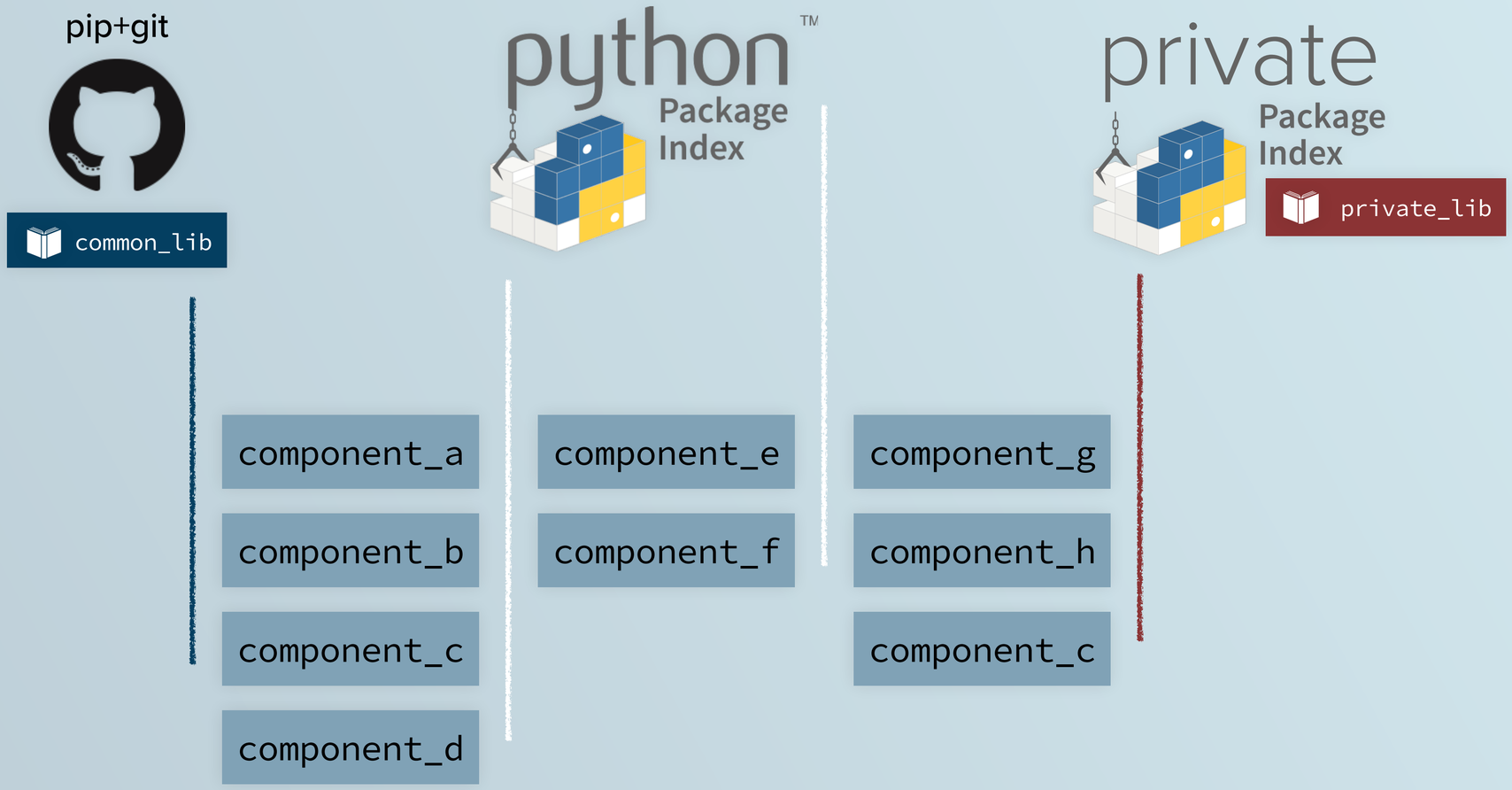
src **wheel** **egg** **git** **zip**

environment separation

venv **pyenv** **pipenv** **virtualenv** **condaenv**

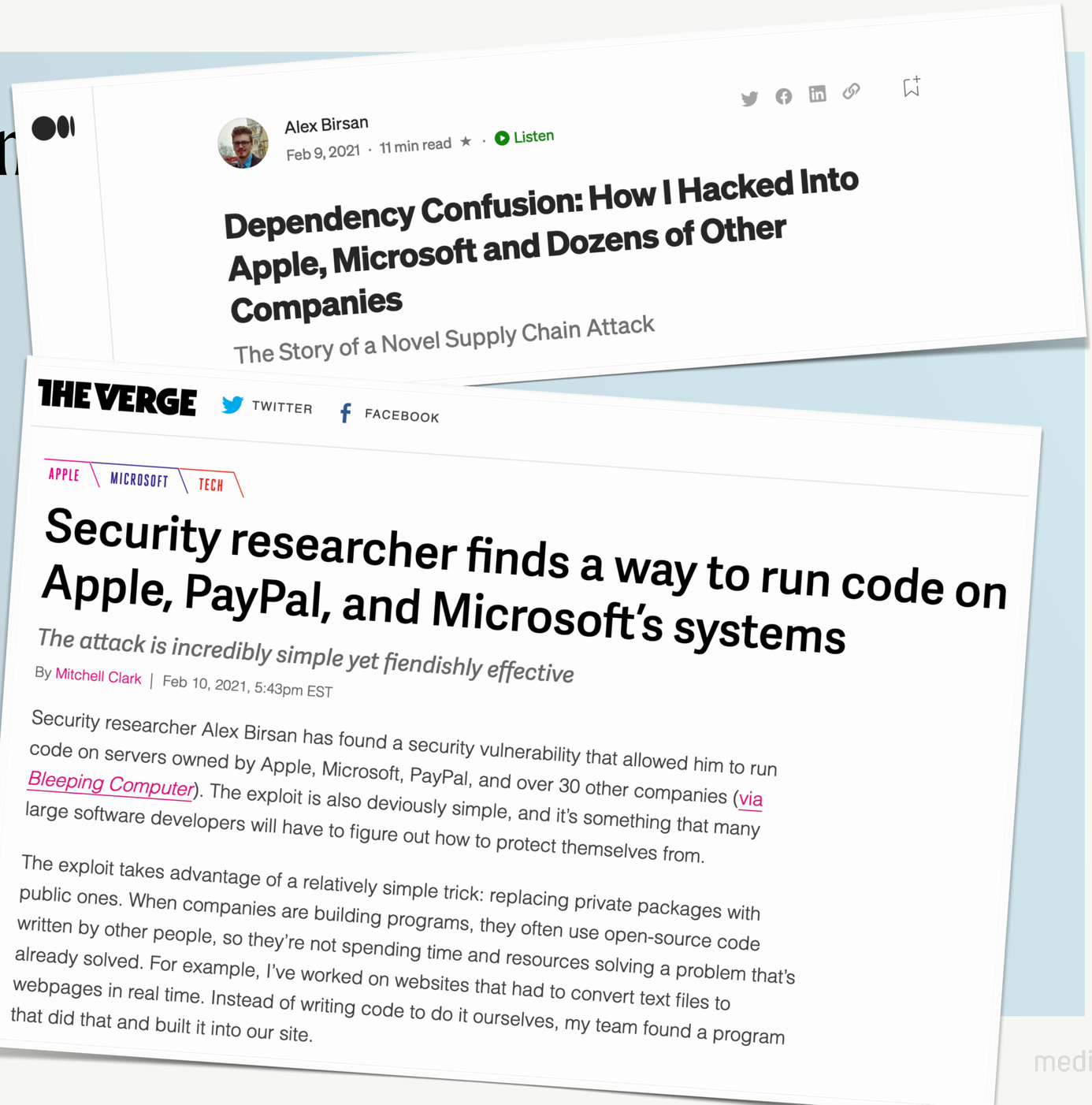

python™
Package Index
common_lib
component_a
component_b
component_c
component_d
component_e
component_f


pip+git
common_lib
python
Package
Index
component_a
component_b
component_c
component_d
component_e
component_f
component_g
component_h
component_c
private_lib


pip+git
common_lib
python™
Package
Index
private
Package
Index
private_lib
component_a
component_b
component_c
component_d
component_e
component_f
component_g
component_h
component_c

private PyPI problem

- authentication
- fast iteration
- single point of failure
- version compatibility
- dependency confusion attack



Alex Birsan
Feb 9, 2021 · 11 min read ★ · Listen

Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies

The Story of a Novel Supply Chain Attack

THE VERGE TWITTER FACEBOOK

APPLE MICROSOFT TECH

Security researcher finds a way to run code on Apple, PayPal, and Microsoft's systems

The attack is incredibly simple yet fiendishly effective

By Mitchell Clark | Feb 10, 2021, 5:43pm EST

Security researcher Alex Birsan has found a security vulnerability that allowed him to run code on servers owned by Apple, Microsoft, PayPal, and over 30 other companies (via *Bleeping Computer*). The exploit is also deviously simple, and it's something that many large software developers will have to figure out how to protect themselves from.

The exploit takes advantage of a relatively simple trick: replacing private packages with public ones. When companies are building programs, they often use open-source code written by other people, so they're not spending time and resources solving a problem that's already solved. For example, I've worked on websites that had to convert text files to webpages in real time. Instead of writing code to do it ourselves, my team found a program that did that and built it into our site.

enter

git submodules

git --dist

About

Documentation

- Reference
- **Book**
- Videos
- External Links

Downloads

Community

This book is available in English.

Full translation available in

azərbaycan dili,

български език,

Deutsch,

...hin it. Perhaps

...developing separately and using in multiple parent

...common issue arises in these scenarios: you want to be able to treat the two projects as separate yet still be able to use one from within the other.

Here's an example. Suppose you're developing a website and creating Atom feeds. Instead of writing your own Atom-generating code, you decide to use a library. You're likely to have to either include this code from a shared library like a CPAN install or Ruby gem, or copy the source code into your own project tree. The issue with including the library is that it's difficult to customize the library in any way and often more difficult to deploy it, because you need to make sure every client has that library available. The issue with copying the code into your own project is that any custom changes you make are difficult to merge when

```
demo/project
$ git submodule init

demo/project
$ git submodule add ../library library_sub
Cloning into 'demo/project/library'...
done.

demo/project
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
 new file:   .gitmodules
 new file:   library_sub

demo/project
$
```

```
demo/project
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
 new file:   .gitmodules
 new file:   library_sub

demo/project
$ cat .gitmodules
[submodule "library"]
 path = library_sub
 url = ../library

demo/project
$
```

```
demo/project
$ cat .gitmodules
[submodule "library"]
 path = library_sub
 url = ../library

demo/project
$ git status
On branch main
  modified:   library_sub
(modified content)

demo/project
$ git status
On branch main
  modified:   library_sub (new
commits)
```

```
demo/project/library_sub
$ touch change.txt

demo/project/library_sub
$ git status
On branch main
Changes to be committed:
  new file:   change.txt

demo/project/library_sub
$ git commit -am "changes"
[main e5a8ae5] world
 1 file changed, 1 insertion(+)
 create mode 100644 change.txt
```

```
demo/project
$ git status
On branch main
  modified:   library_sub (new
commits)

demo/project
$ git commit -am "sub changes"
[main 2f62b1e] sub changes
1 file changed, 1 insertion(+),
1 deletion(-)

demo/project
$
```

```
demo/project/library_sub
$ git commit -am "changes"
[main e5a8ae5] changes
 1 file changed, 1 insertion(+)
 create mode 100644 change.txt

demo/project/library_sub
$
```

```
demo/project/library_sub
$ git commit -am "changes"
[main e5a8ae5] changes
 1 file changed, 1 insertion(+)
 create mode 100644 change.txt

demo/project/library_sub
$
```

```
demo/project/library_sub
$ git commit -am "changes"
[main e5a8ae5] changes
 1 file changed, 1 insertion(+)
 create mode 100644 change.txt

demo/project/library_sub
$
```

```
demo/project/library_sub
$ git commit -am "changes"
[main e5a8ae5] changes
 1 file changed, 1 insertion(+)
 create mode 100644 change.txt

demo/project/library_sub
$ git checkout --branch feat
Switched to a new branch 'feat'

demo/project/library_sub
$ touch feat.txt

demo/project/library_sub
$ git commit -am "feature"
[feat e2ac8afa] changes
 1 file changed, 1 insertion(+)
 create mode 100644 feat.txt
```

```
demo/library
$ ls



demo/library
$ git branch
* main

demo/library
$
```

```
demo/project/library_sub
$ git commit -am "feature"
[feat e2ac8afa] changes
 1 file changed, 1 insertion(+)
 create mode 100644 feat.txt

demo/project/library_sub
$ git push --set-upstream
origin feat
Writing objects: 100% (3/3)
Total 3, reused 0, pack-reused 0
To /demo/library
 * [new branch]      feat -> feat
Branch 'feat' set up to track
remote branch 'feat' from
'origin'.
```

```
demo/library
$ git branch
* main



demo/library
$ git branch
* main
  branch

demo/library
$
```

```
Writing objects: 100% (3/3)
Total 3, reused 0, pack-reused 0
To /demo/library
 * [new branch]      feat -> feat
Branch 'feat' set up to track
remote branch 'feat' from
'origin'.

demo/project/library_sub
$ cd ..

demo/project
$ git branch
* main

demo/project
$
```

```
demo/library
$ git branch
* main
  branch



demo/library
$
```

project setup

```
project/
    README.md
    Dockerfile
    .gitignore
    requirements.txt
    project/
        __init__.py
        …
```

```
library/
    README.md
    Dockerfile
    .gitignore
    requirements.txt
    library/
        __init__.py
        …
```

project setup

```
$ git submodule add git@gitlab.com/library library
$ export PYTHONPATH=./library/:${PYTHONPATH}
```

```
project/
  …
  project/
    __init__.py
    …
  library/
    …
    library/
      __init__.py
      …
```

```
$ git submodule add git@gitlab.com/library _library
$ ln -s _library/library library
```

```
project/
  …
  project/
    __init__.py
    …
  _library/
    …
    library/
      __init__.py
      …
  library/ 🔗
```

point to branches/tags

```
project  DEV-2342
    common_lib_1  v4.2.23
    common_lib_2  v0.23.42
    common_lib_3  DEV-2342
```

multiple states at the same time

```
project_a  main
    common_lib  v2.3.42
```

```
project_b  main
    common_lib  v4.2.23
```

IDE support/import discovery

```
project/
  project/
    _library/
      library/
    library/
```

(external) dependency resolution

- concat requirements.txt
- ordering matters

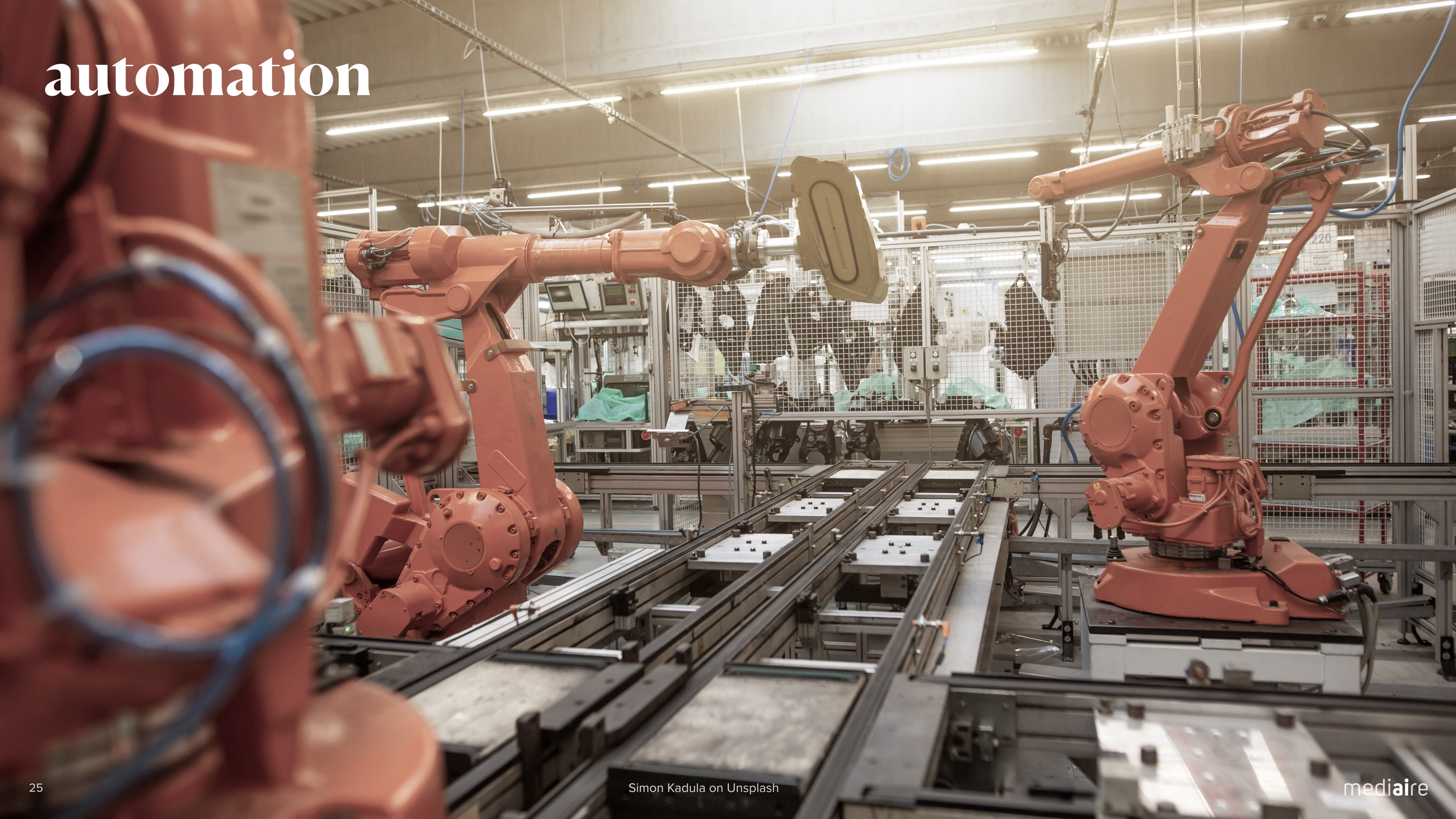
duplication

- disk space
- compile time
- build time

workflow

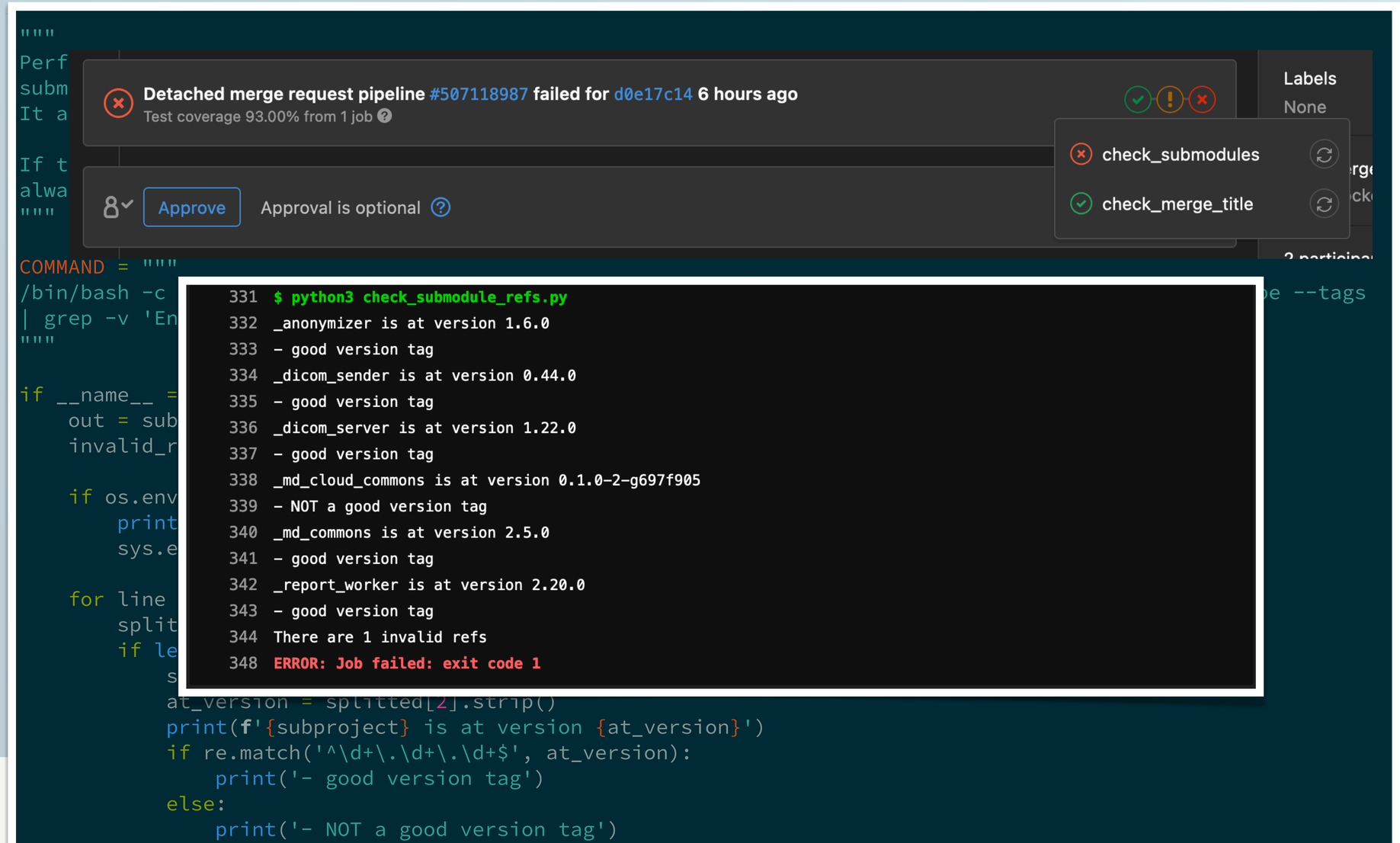
- learning curve
- git submodule init/update
- commit submodules in untagged state

automation

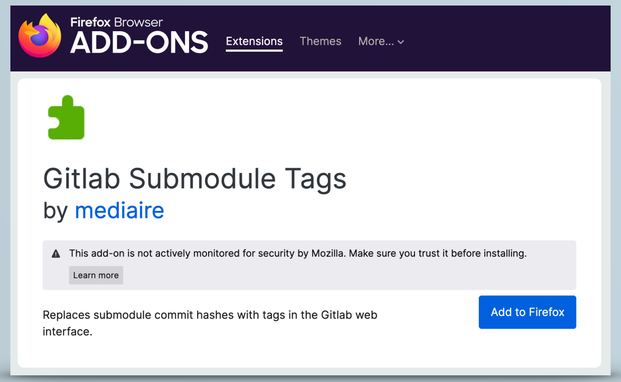


Simon Kadula on Unsplash

prevent untagged submodules on CI



Gitlab UI



M

md_cloud_connector

Project ID: Leave project

Unstar 1 Fork 0

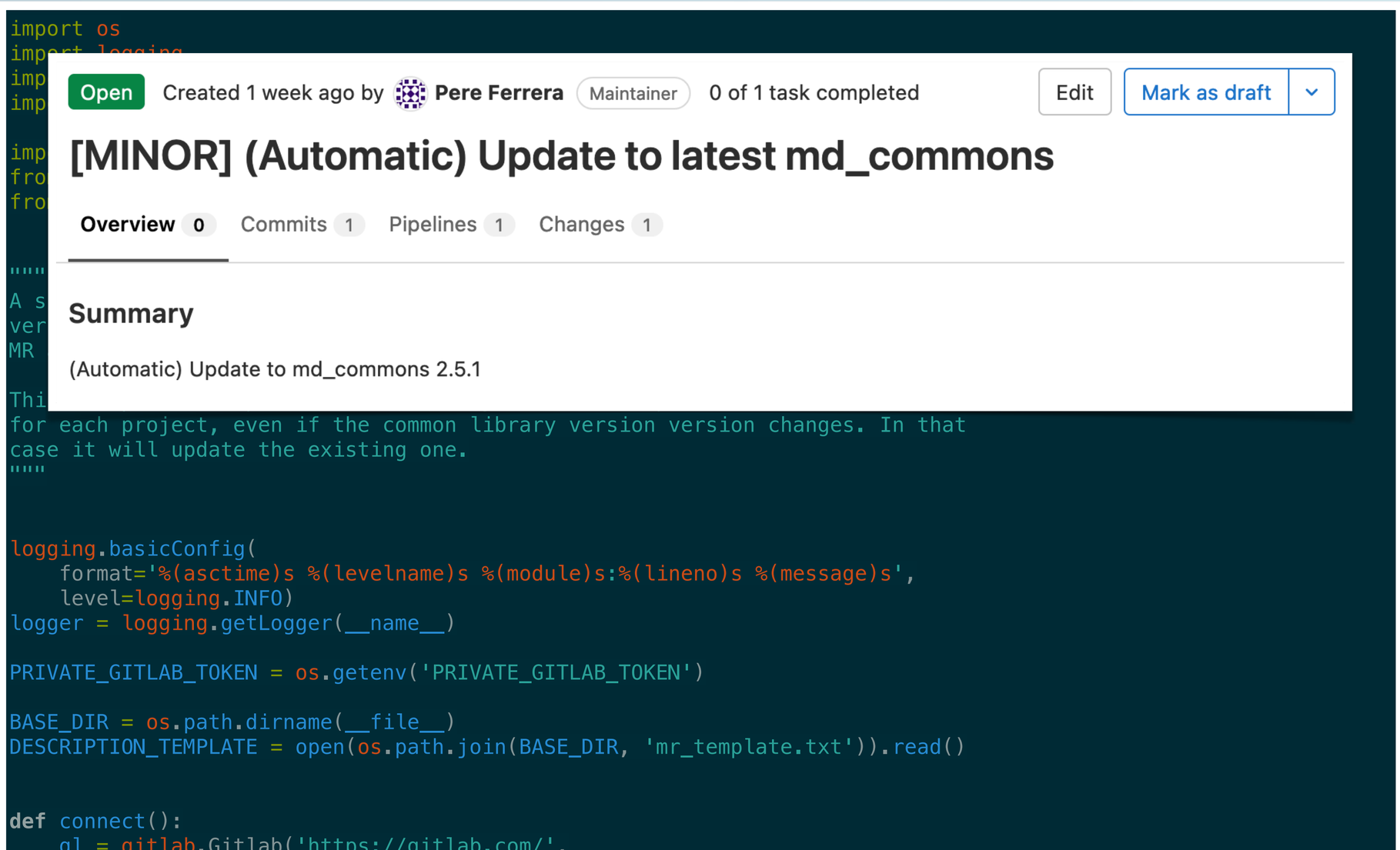
84 Commits **5** Branches **25** Tags **1.9 MB** Files **146 MB** Storage

main md_cloud_connector / + History Find file Web IDE Clone

Version 0.8.0, automatic version bump
automatic_version_bot authored 1 month ago
247ef127

Name	Last commit	Last update
config	[PATCH] BS-92: md_cloud_connector: Im...	3 months ago
tests	[MINOR] BS-37 md_cloud_connector: Br...	1 month ago
anonymizer @ 1.6.0	[MINOR] [BS-106] Update anonymizer	3 months ago
dicom_sender @ 0.44.0	[MINOR] DEV-2396: Production-ready m...	5 months ago
dicom_server @ 1.22.0	[PATCH] BS-55: md_cloud_connector: cl...	4 months ago
md_cloud_commons @ 0.23.1	[PATCH] [BS-79] Amend #4: We cannot r...	3 months ago
md_commons @ 2.5.0	[PATCH] [BS-79] Amend #4: We cannot r...	3 months ago
mediaire_toolbox @ 1.20.1	[MINOR] BS-16: Compiled release image ...	6 months ago
report_worker @ 1.14.2	[PATCH] BS-92: md_cloud_connector: Im...	3 months ago

automatically create library update MRs



type checking

```
# .mypy.ini
[mypy]
ignore_errors = True

[mypy-my_project_dir.*]
ignore_errors = False
```

Summary

- fast iteration
- simple setup
- de-coupled across projects

outlook

monorepo

submodule
subtree
worktree

Matthew Ansley on Unsplash

thanks, connect, src

mediaire

github.com/phistep

philippstephan.de
/blog/posts/pycon22-talk/

www.mediaire.de

mediaire.jobs.personio.de

github.com/mediaire
/submodule-dependencies