

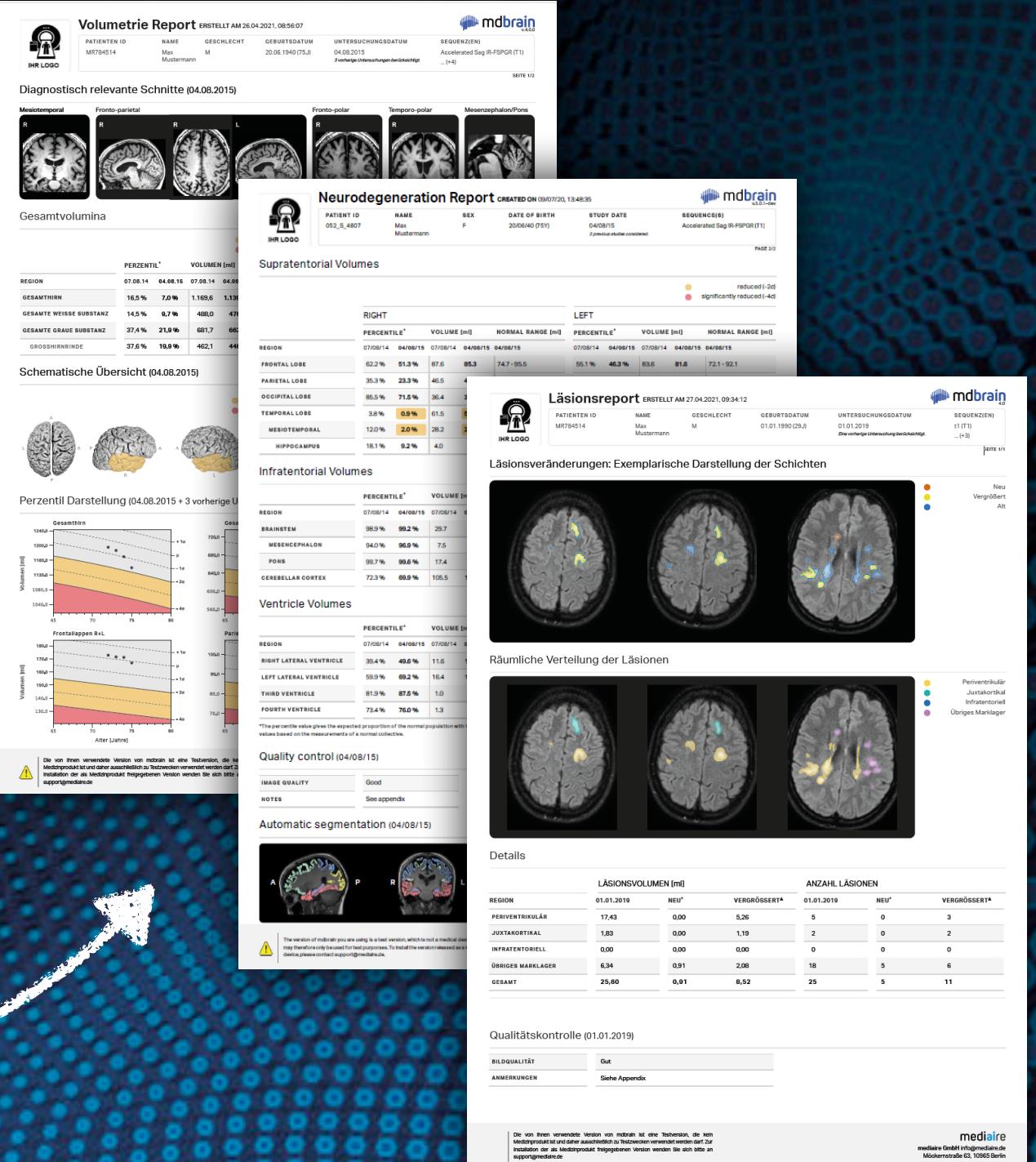
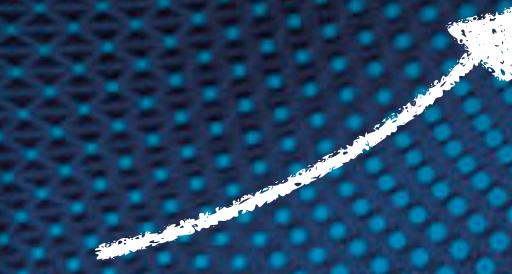
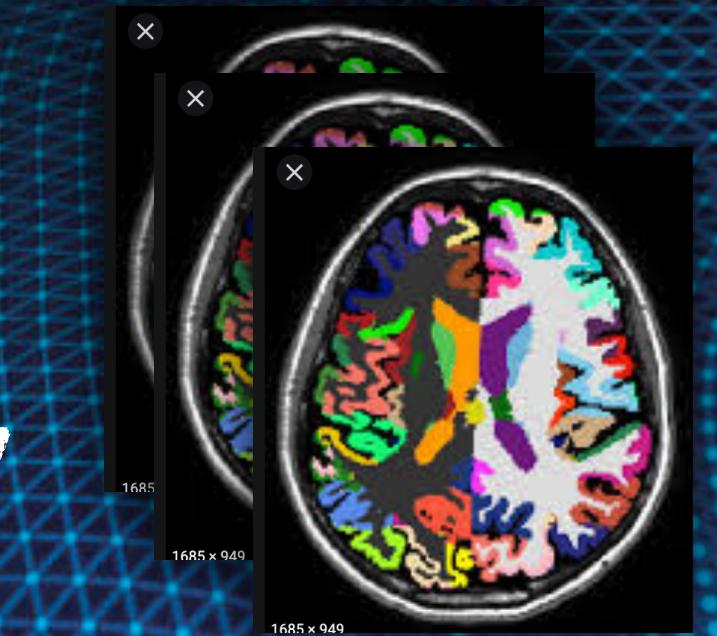
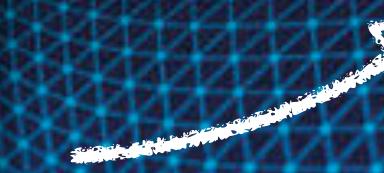
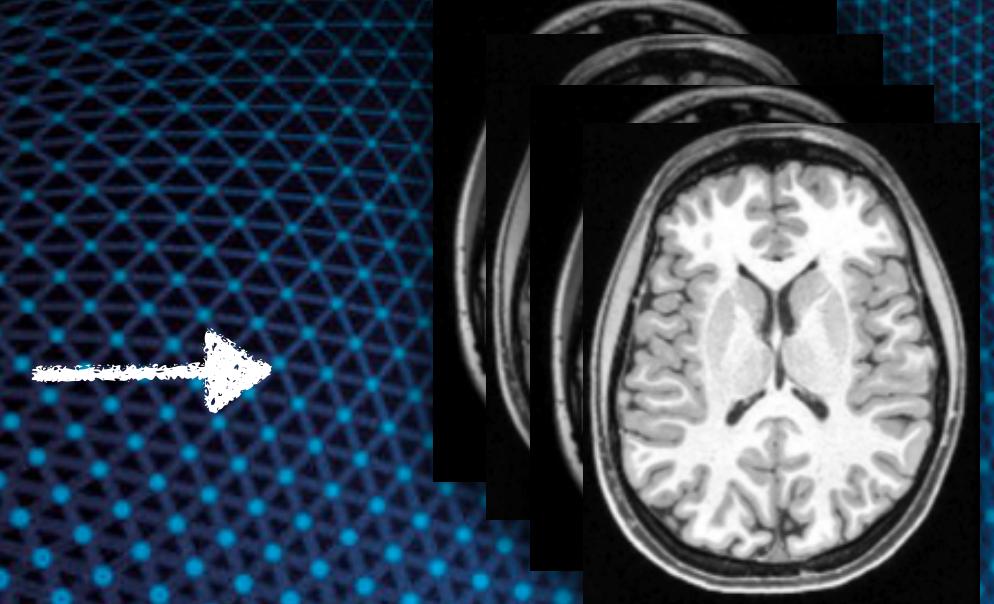
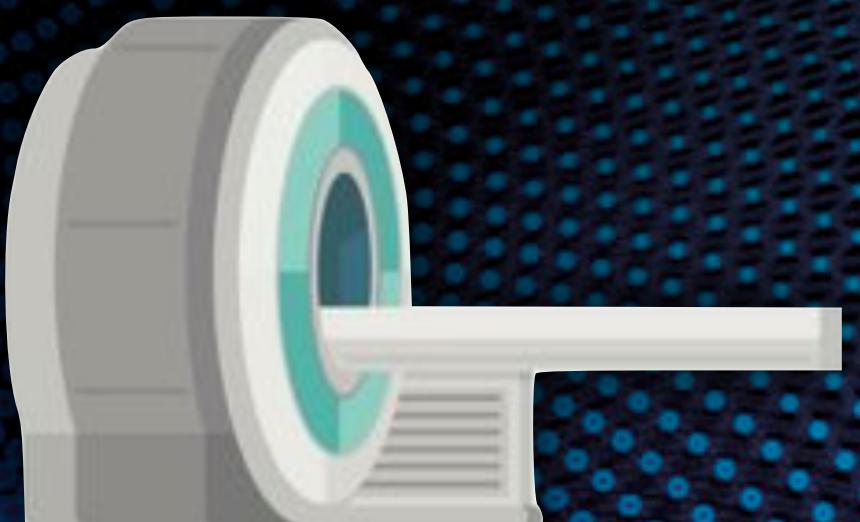
Quitting pip:

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

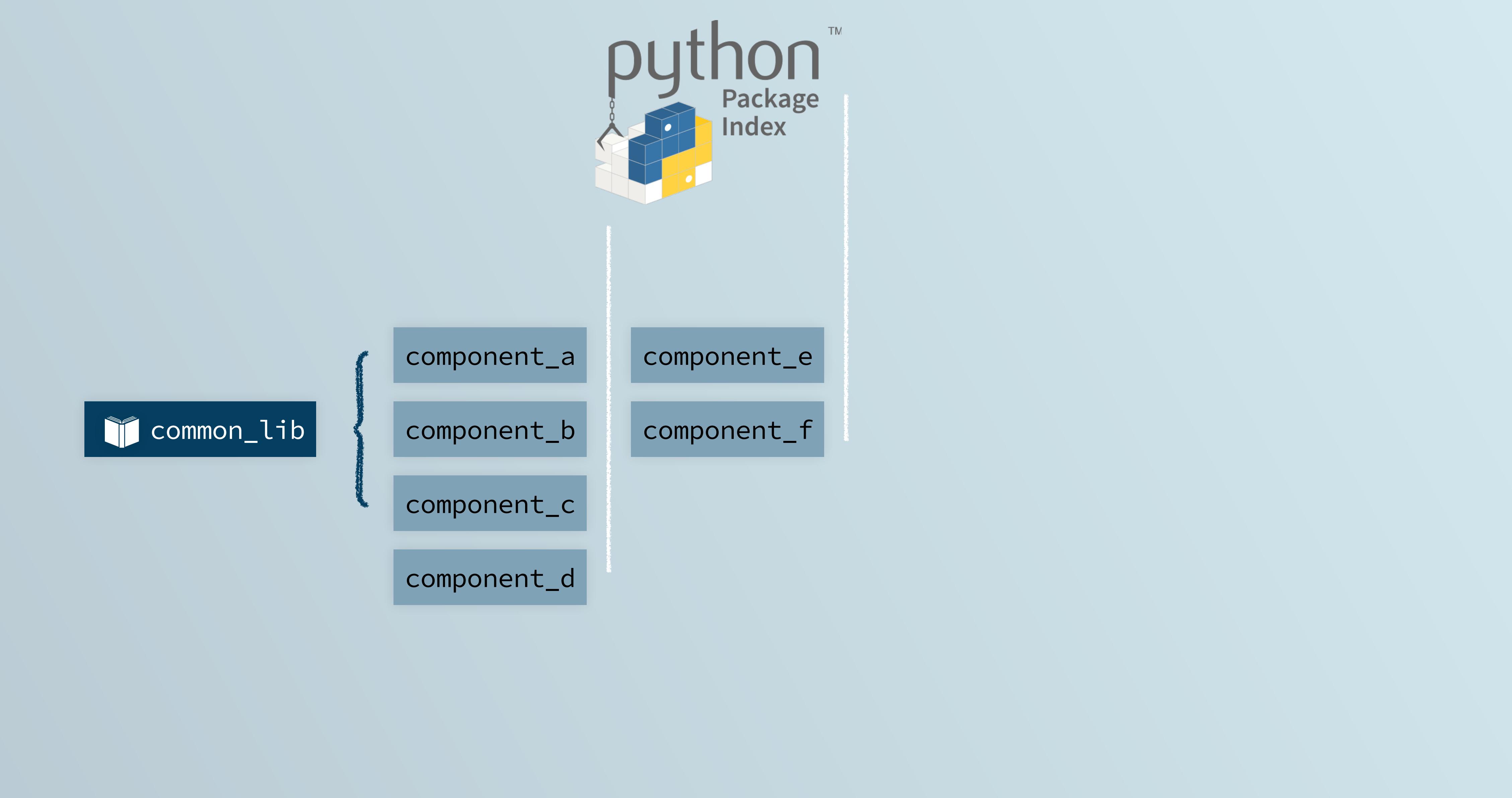
component_c

component_d

numpy

sqlalchemy

matplotlib



The Zen of Python

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

```
>>> import this
```

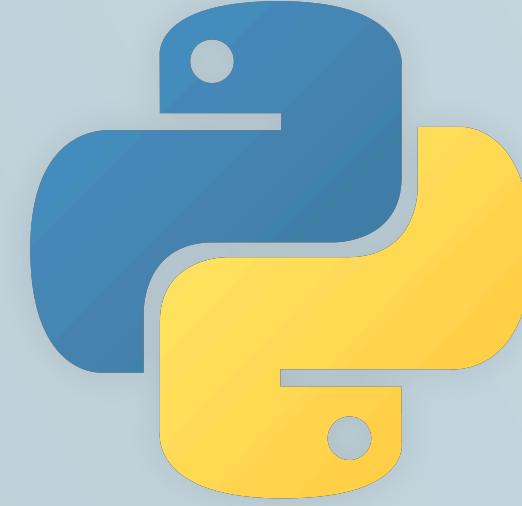
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
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!

package managers



`easy_install`
pip



Conda



Poetry



package formats

src

wheel

egg

git

zip

environment separation

venv

pyenv

pipenv

virtualenv

condaenv



{

component_a
component_b
component_c
component_d



component_e
component_f

pip+git



 common_lib

component_a

component_b

component_c

component_d

python™
Package
Index



component_e

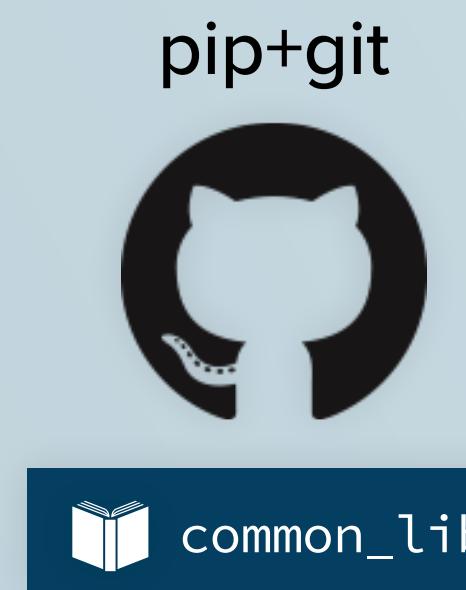
component_f

component_g

component_h

component_c

 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

A screenshot of a news article from The Verge. The article is titled "Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies" by Alex Birsan, published on Feb 9, 2021. The article discusses a novel supply chain attack. The Verge logo is at the top left, along with social media links for Twitter, Facebook, and LinkedIn. Below the main title is a subtitle: "The Story of a Novel Supply Chain Attack". The article is categorized under "APPLE", "MICROSOFT", and "TECH". The main headline reads: "Security researcher finds a way to run code on Apple, PayPal, and Microsoft's systems". A sub-headline states: "The attack is incredibly simple yet fiendishly effective". The author is Mitchell Clark, and the date is Feb 10, 2021, 5:43pm EST. The article text explains that security researcher Alex Birsan found a vulnerability allowing him to run code on servers owned by Apple, Microsoft, PayPal, and over 30 other companies (via Bleeping Computer). The exploit is simple and affects large software developers. It then goes into detail about how the exploit works, mentioning replacing private packages with public ones and leveraging open-source code.

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

THE VERGE

TWITTER FACEBOOK

APPLE MICROSOFT TECH

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

The Story of a Novel Supply Chain Attack

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



The image is a meme from Meme-Generator.com. It features a man with braided hair laughing heartily. The text 'Yo dawg, I heard you like git' is at the top, and 'So I put git in your git so you can commit while you commit' is at the bottom. The source 'Meme-Generator.com' is visible in the bottom right corner of the image.

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](#).

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

...

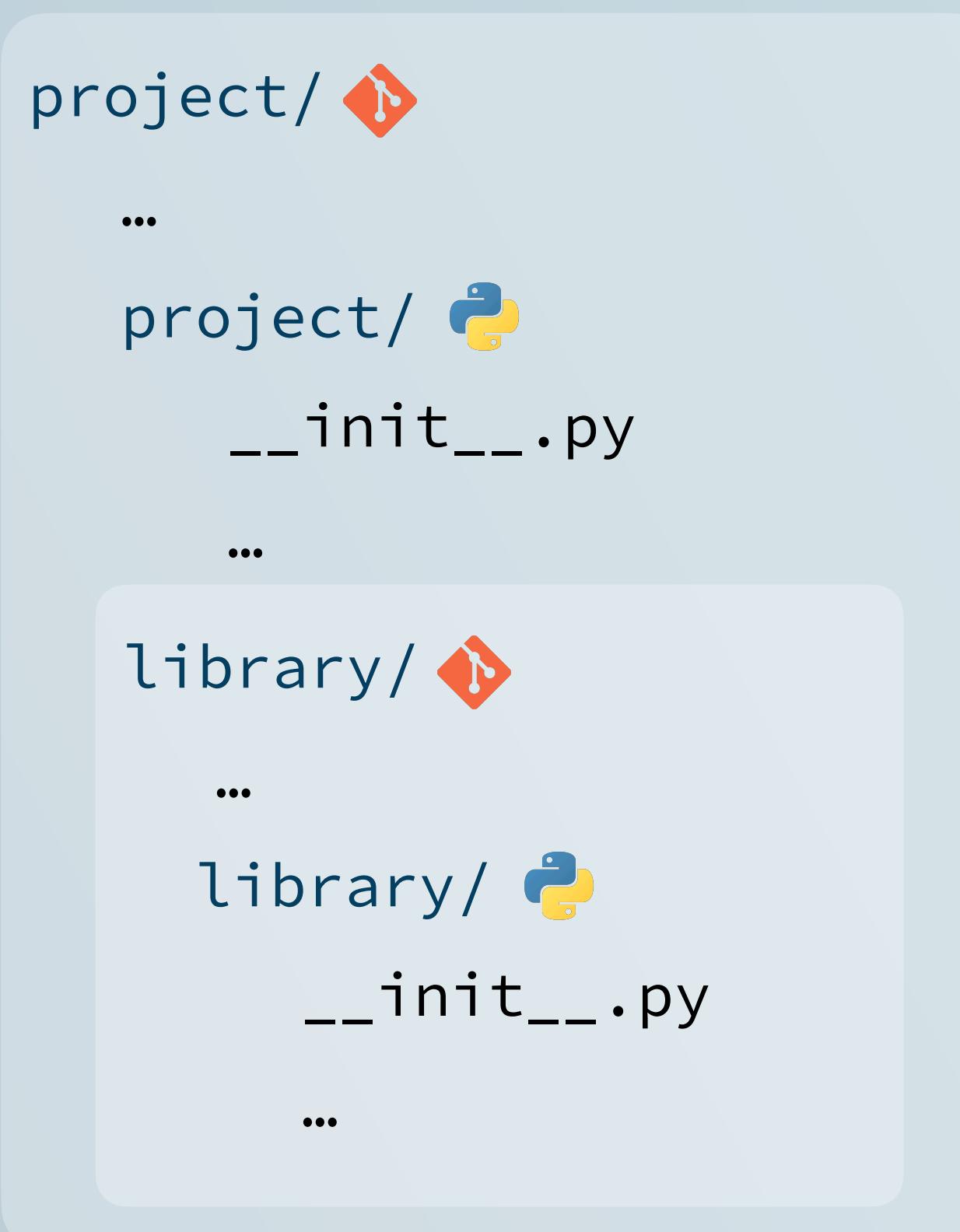
library/ 🛠

README.md
Dockerfile
.gitignore
requirements.txt
library/ 🐍
__init__.py

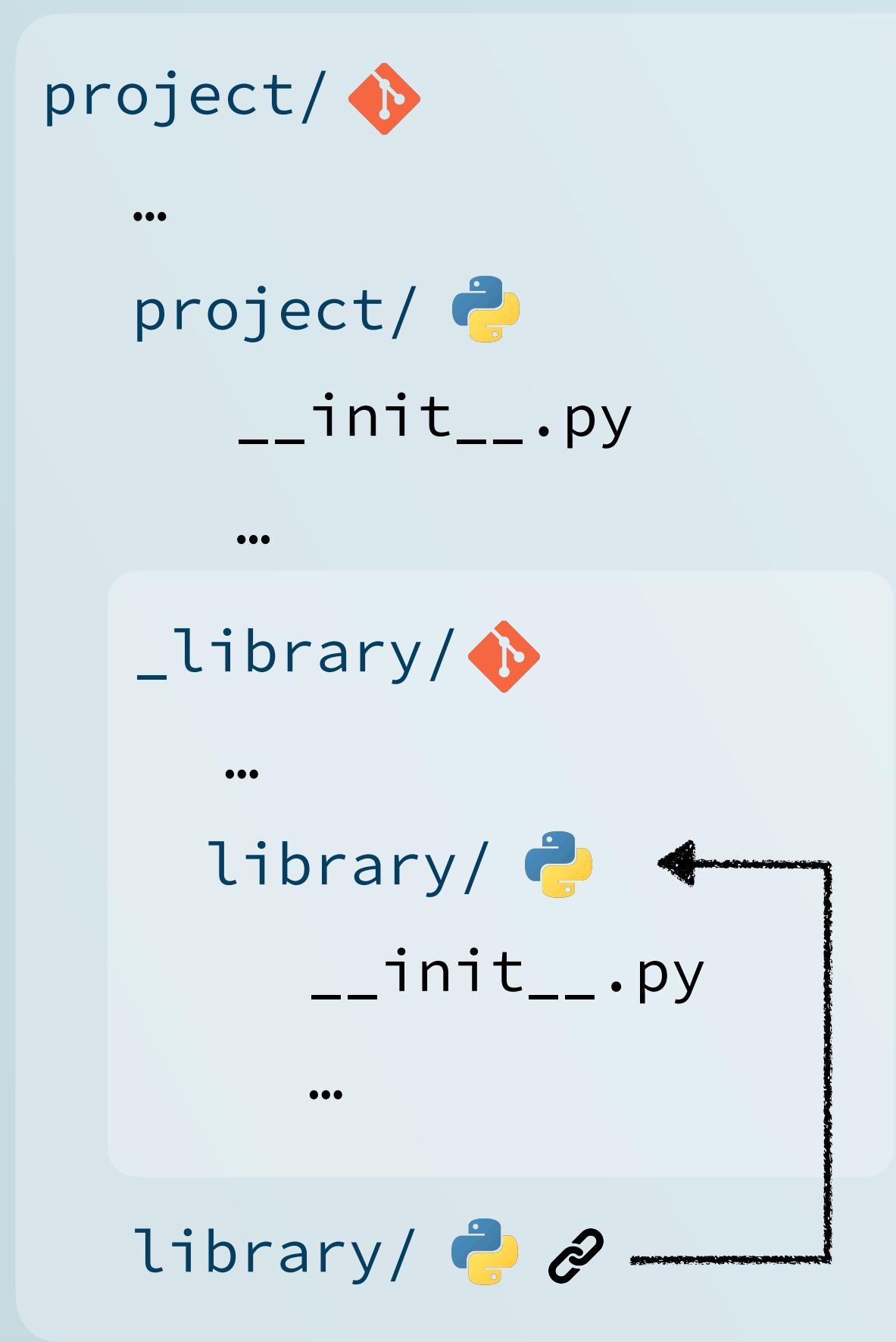
...

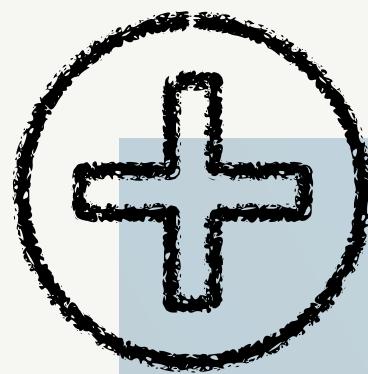
project setup

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



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





point to branches/tags

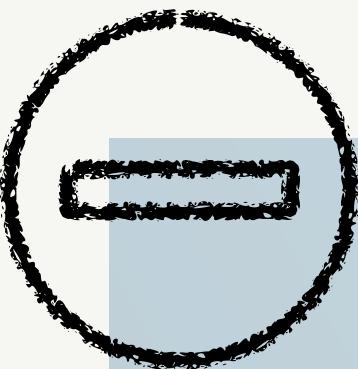
```
project ↗ DEV-2342  
common_lib_1 🏷 v4.2.23  
common_lib_2 🏷 v0.23.42  
common_lib_3 ↗ DEV-2342
```

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



prevent untagged submodules on CI

The screenshot shows a GitLab merge request pipeline interface. At the top, there is a status message: "Detached merge request pipeline #507118987 failed for d0e17c14 6 hours ago". Below this, there is an "Approve" button with a dropdown menu showing "8" and "Approval is optional". To the right, there is a "Labels" section with "None" and a "check_submodules" badge. A command-line log window is open, showing the output of a script named `check_submodule_refs.py`. The log output includes several good version tags (e.g., `_anonymizer`, `_dicom_sender`, `_dicom_server`) and one bad version tag (`_md_cloud_commons`), which is noted as "NOT a good version tag". The log concludes with an error message: "ERROR: Job failed: exit code 1".

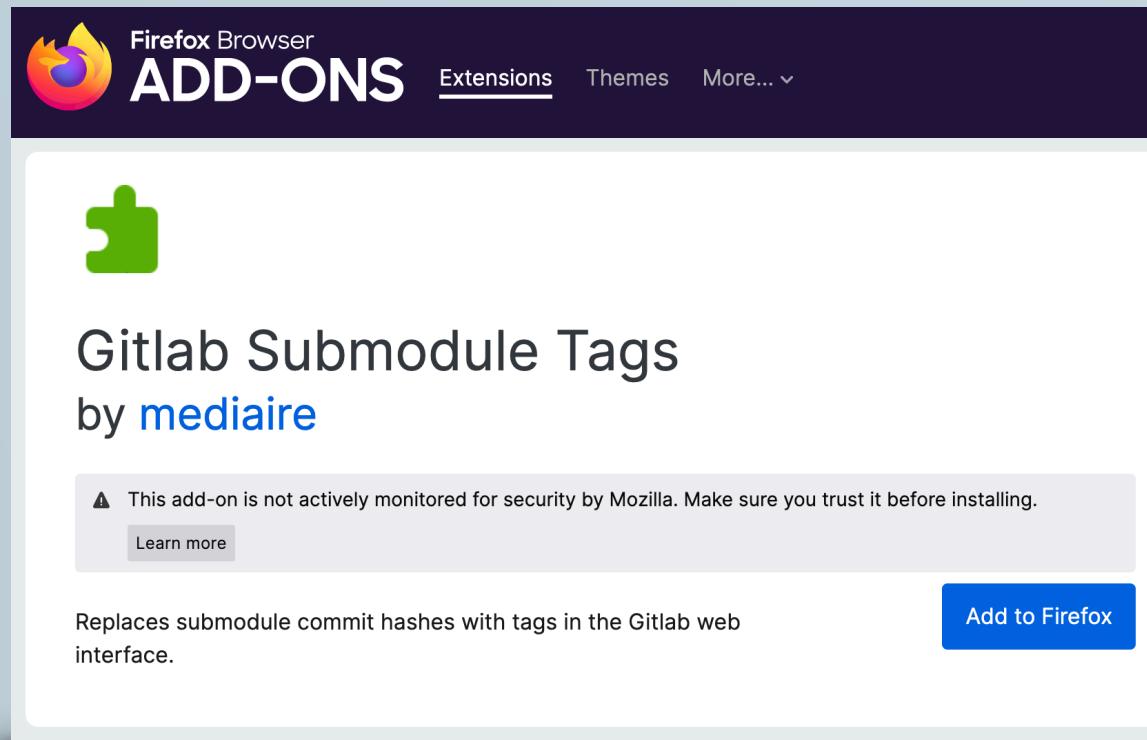
```
"""
Perf
subm
It a
If t
alwa
"""
COMMAND = """
/bin/bash -c
| grep -v 'En
"""

if __name__ =
    out = sub
    invalid_r
    if os.env
        print
        sys.e
    for line
        split
        if le
            at_version = splitted[2].strip()
            print(f'{subproject} is at version {at_version}')
            if re.match('^\d+\.\d+\.\d+$', at_version):
                print('- good version tag')
            else:
                print('- NOT a good version tag')
"""

$ python3 check_submodule_refs.py
_anonymizer is at version 1.6.0
- good version tag
_dicom_sender is at version 0.44.0
- good version tag
_dicom_server is at version 1.22.0
- good version tag
_md_cloud_commons is at version 0.1.0-2-g697f905
- NOT a good version tag
_md_commons is at version 2.5.0
- good version tag
_report_worker is at version 2.20.0
- good version tag
There are 1 invalid refs
ERROR: Job failed: exit code 1

```

Gitlab UI



The screenshot shows the Gitlab UI for the project 'md_cloud_connector'. The top navigation bar includes a user icon, the project name, a lock icon, a 'Leave project' button, and project statistics: 84 Commits, 5 Branches, 25 Tags, 1.9 MB Files, and 146 MB Storage. Below the stats is a navigation bar with dropdowns for 'main' and 'md_cloud_connector /', a '+' button, and links for History, Find file, Web IDE, and Clone. A commit card for 'Version 0.8.0, automatic version bump' by 'automatic_version_bot' is shown, along with the commit hash '247ef127'. The main content area displays a table of submodule tags with columns for Name, Last commit, and Last update. The submodules listed are config, tests, anonymizer @ 1.6.0, dicom_sender @ 0.44.0, dicom_server @ 1.22.0, md_cloud_commons @ 0.23.1, md_commons @ 2.5.0, mediaire_toolbox @ 1.20.1, and report_worker @ 1.14.2.

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

The screenshot shows a GitLab merge request interface. At the top, there's a code snippet fragment:

```
import os
import logging
imp
imp
imp
fro
fro
.....
A s
ver
MR
This
for each project, even if the common library version version changes. In that
case it will update the existing one.
.....
```

Below the code, the merge request details are shown:

[MINOR] (Automatic) Update to latest md_commons

Created 1 week ago by **Pere Ferrera** Maintainer 0 of 1 task completed

Buttons: **Edit**, **Mark as draft**

Navigation tabs: Overview 0, Commits 1, Pipelines 1, Changes 1

Summary

(Automatic) Update to md_commons 2.5.1

This

for each project, even if the common library version version changes. In that case it will update the existing one.

.....

logging.basicConfig(
 format='%(asctime)s %(levelname)s %(module)s:%(lineno)s %(message)s',
 level=logging.INFO)
logger = logging.getLogger(__name__)

PRIVATE_GITLAB_TOKEN = os.getenv('PRIVATE_GITLAB_TOKEN')

BASE_DIR = os.path.dirname(__file__)
DESCRIPTION_TEMPLATE = open(os.path.join(BASE_DIR, 'mr_template.txt')).read()

def connect():
 gl = gitlab.Gitlab('https://gitlab.com/')

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
worktree**

Matthew Ansley on Unsplash

thanks, connect, src



github.com/phistep
[philippstephan.de
/blog/posts/pycon22-talk/](https://philippstephan.de/blog/posts/pycon22-talk/)

mediaire

www.mediaire.de
mediaire.jobs.personio.de



[github.com/mediaire
/submodule-dependencies](https://github.com/mediaire/submodule-dependencies)