

Gérez vos projects CMake et leurs dépendances avec pip

2022-11-20

Guilhem Saurel

Available at

<https://nim65s.github.io/talks/cmake-wheel.pdf>

Under License



<https://creativecommons.org/licenses/by-sa/4.0/>

Source

[https://github.com/nim65s/talks :
cmake-wheel.md](https://github.com/nim65s/talks : cmake-wheel.md)

Discussions

<https://matrix.to/#/room/#cmake-wheel:matrix.org>

- C / C++ (et plus si affinités)
- GCC / LLVM / MSVC / Intel / ...
- Linux / *BSD / macOS / Windows / ...



Figure 1: CMake

```
find_package(Boost REQUIRED COMPONENTS regex)  
add_library(pipo SHARED pipo.hpp pipo.cpp)  
include_directories(${Boost_INCLUDE_DIRS})  
target_link_libraries(pipo PUBLIC  
    ${Boost_LIBRARIES})
```

```
find_package(Boost REQUIRED COMPONENTS regex)
add_library(pipo SHARED pipo.hpp pipo.cpp)
include_directories(${Boost_INCLUDE_DIRS})
target_link_libraries(pipo PUBLIC
    ${Boost_LIBRARIES})

-include_directories(${Boost_INCLUDE_DIRS})
-target_link_libraries(pipo PUBLIC
-    ${Boost_LIBRARIES})
+target_link_libraries(pipo PUBLIC Boost::regex)
```

```
find_package(Boost REQUIRED COMPONENTS regex)
add_library(pipo SHARED pipo.hpp pipo.cpp)
include_directories(${Boost_INCLUDE_DIRS})
target_link_libraries(pipo PUBLIC
    ${Boost_LIBRARIES})

-include_directories(${Boost_INCLUDE_DIRS})
-target_link_libraries(pipo PUBLIC
-    ${Boost_LIBRARIES})
+target_link_libraries(pipo PUBLIC Boost::regex)
```

=> Paquets relocalisables

```
python -m pip install --user django
```

A build-system independent format for source trees

Permet à n'importe qui de créer un système de build en écrivant une fonction:

```
def build_wheel(  
    wheel_directory,  
    config_settings=None,  
    metadata_directory=None,  
):  
    ...
```

```
from subprocess import run

def build_wheel(...):
    run("cmake -S src -B build")
    run("cmake --build build")
    run("cmake --build build -t test")
    run("cmake --install build")
```

A wheel is a ZIP-format archive with a specially formatted file name and the .whl extension.

A wheel is a ZIP-format archive with a specially formatted file name and the .whl extension.

eg: `hpy_fcl-2.1.3-4-cp310-cp310-
manylinux_2_28_x86_64.whl`

[project]

```
name = "cmeel-example"  
version = "0.2.3"  
description = "Example project"  
readme = "README.md"  
license = "BSD-2-Clause"  
authors = ["guilhem.saurel@laas.fr"]
```

[project.urls]

```
repository = "https://github.com/cmake-wheel/cmeel-e"
```

[build-system]

```
requires = ["cmeel[build]"]  
build-backend = "cmeel.build"
```

- `python -m pip install \`
`git+https://github.com/cmake-wheel/cmeel-`
`example.git`
- `python -m pip install cmeel-example`

- `python -m pip install \`
`git+https://github.com/cmake-wheel/cmeel-`
`example.git`
- `python -m pip install cmeel-example`
- `cmeel-add 3 4`
- `python -c "import cmeel_example;`
`print(cmeel_example.cmeel_add(3, 4))"`

with cibuildwheel and github actions for:

- CPython 3.7 - 3.11 / pypy 3.7 - 3.9
- manylinux / musllinux / macos
- x86_64 / aarch64 / ppc64le / i686 / s390x

And cirrus-CI for Apple Silicon

- -
DCMAKE_INSTALL_PREFIX=\${SITELIB}/cmeel.prefix
- \${SITELIB}/cmeel.pth
- .../cmeel.prefix/bin=> cmeel.run:cmeel_run
- helpers for \$CMAKE_PREFIX_PATH, \$LD_LIBRARY_PATH,
...

- -
DCMAKE_INSTALL_PREFIX=\${SITELIB}/cmeel.prefix
- \${SITELIB}/cmeel.pth
- .../cmeel.prefix/bin=> cmeel.run:cmeel_run
- helpers for \$CMAKE_PREFIX_PATH, \$LD_LIBRARY_PATH,
...

Contraintes:

- CMake
- paquets relocatables
- RPATH/@loader_path

- <https://github.com/cmake-wheel/cmeel>
- <https://github.com/cmake-wheel/cmeel-example>
- <https://cmeel.rtf.d.io>

La parole est à vous :)