

Build & Packaging

Université Toulouse Paul Sabatier - KEAT9AA1

2024-09-12

Guilhem Saurel

Available at

**[https://homepages.laas.fr/gsaurel/teach/
2024-2025/M2_ISTR/6-build-packaging.pdf](https://homepages.laas.fr/gsaurel/teach/2024-2025/M2_ISTR/6-build-packaging.pdf)**

Under License



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

Source

**<https://gitlab.laas.fr/gsaurel/teach> :
2024-2025/M2_ISTR/6-build-packaging.md**

Contact

Matrix: @gsaurel:laas.fr
Mail: gsaurel@laas.fr

Introduction

- 1 understanding:
 - process execution
 - scripts, binaries, and python modules
- 2 knowledge:
 - distribute software
 - sources and binaries
- 3 advices:
 - FHS
 - software management on a system

- 1 linux
- 2 macos, *BSD
- 3 windows

- 1 linux
- 2 macos, *BSD
- 3 windows
 - with CMake
 - with WSL

Part 1: Understanding execution


```
gsaurel@linux ~$ ls -l ex  
-rwxr-xr-x [...] my-script
```

```
gsaurel@linux ~$ cat ex/my-script  
#!/bin/bash  
echo hello
```

```
gsaurel@linux ~$ ls -l ex  
-rwxr-xr-x [...] my-script
```

```
gsaurel@linux ~$ cat ex/my-script  
#!/bin/bash  
echo hello
```

```
gsaurel@linux ~$ /home/gsaurel/ex/my-script  
hello
```

```
gsaurel@linux ~$ ls -l ex  
-rwxr-xr-x [...] my-script
```

```
gsaurel@linux ~$ cat ex/my-script  
#!/bin/bash  
echo hello
```

```
gsaurel@linux ~$ /home/gsaurel/ex/my-script  
hello
```

```
gsaurel@linux ~$ ./ex/my-script  
hello
```

```
gsaurel@linux ~$ echo $PATH  
/usr/local/bin:/usr/bin:/bin
```

```
gsaurel@linux ~$ echo $PATH  
/usr/local/bin:/usr/bin:/bin
```

```
gsaurel@linux ~$ export PATH=/home/gsaurel/ex:$PATH  
gsaurel@linux ~$ my-script  
hello
```

```
#include <iostream>

int main() {
    std::cout << "hello" << std::endl;
    return 0;
}
```

```
gsaurel@linux ~/ex$ g++ -o my-bin main.cpp  
gsaurel@linux ~/ex$ my-bin  
hello
```

```
gsaurel@linux ~/ex$ g++ -o my-bin main.cpp
gsaurel@linux ~/ex$ my-bin
hello
```

```
gsaurel@linux ~/ex$ ls -l
-rw-r--r-- [...] main.cpp
-rwxr-xr-x [...] my-bin
-rwxr-xr-x [...] my-script
```



```
gsaurel@linux ~/ex$ ldd my-bin
linux-vdso.so.1 (0x00007ffffb5ff100)
/lib/x86_64-linux-gnu/libnss_sss.so.2 (0x0000722d90bce000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x0000722d90bce000)
libc.so.6 => /usr/lib/x86_64-linux-gnu/libc.so.6 (0x0000722d90bce000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x0000722d90bce000)
/lib64/ld-linux-x86-64.so.2 (0x0000722d90bce000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1
```

```
#include <iostream>
#include <cpr/cpr.h>

int main() {
    cpr::Url url{"https://www.laas.fr/fr/"};
    cpr::Response r = cpr::Get(url);
    std::cout << r.text << std::endl;
    return 0;
}
```

```
gsaurel@linux ~/ex$ g++ -o my-bin main.cpp
/usr/bin/ld : /tmp/ccAcavy5.o : dans la fonction « cp
main.cpp:(.text._ZN3cpr3GetIJRNS_3UrLEEEENS_8Response
/usr/bin/ld : main.cpp:(.text._ZN3cpr3GetIJRNS_3UrLEE
/usr/bin/ld : /tmp/ccAcavy5.o : dans la fonction « vo
main.cpp:(.text._ZN3cpr4priv19set_option_internalILb0
/usr/bin/ld : /tmp/ccAcavy5.o : dans la fonction « st
main.cpp:(.text._ZNSt4pairIKNSt7__cxx1112basic_string
collect2: erreur: ld a retourné le statut de sortie 1
```

```
gsaurel@linux ~/ex$ g++ -o my-bin -l cpr main.cpp
gsaurel@linux ~/ex$ my-bin
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>
    Accueil - LAAS-CNRS
[...]
```

To compile, **my-bin** require:

- **/usr/lib/libcpr.so**
- **/usr/include/cpr/cpr.h**

To compile, **my-bin** require:

- **/usr/lib/libcpr.so**
- **/usr/include/cpr/cpr.h**

Or add:

- **g++ -L ...**, if **libcpr.so** is elsewhere
- **g++ -I ...**, if **cpr/cpr.h** is elsewhere

To run, **my-bin** require:

- **`/usr/lib/libcpr.so`**

To run, **my-bin** require:

- **/usr/lib/libcpr.so**

If **libcpr.so** is elsewhere, one can:

- set **\$LD_LIBRARY_PATH** environment variable
- set **RPATH** attribute inside **my-bin** binary


```
gsaurel@linux ~/ex$ ldd my-bin
linux-vdso.so.1 (0x00007fff9188b000)
libcpr.so.1 => /usr/lib/libcpr.so.1 (0x00007217eb
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x0000
[...]
```

either:

- `my_module.py`
- `my_module.so`
- `my_module/__init__.py`

either:

- `my_module.py`
- `my_module.so`
- `my_module/__init__.py`

In a folder in the **`sys.path`** list, modifiable through:

- virtual environments
- **`$PYTHONPATH`** environment variable
- **`.pth`** file

either:

- `my_module.py`
- `my_module.so`
- `my_module/__init__.py`

In a folder in the **`sys.path`** list, modifiable through:

- virtual environments
- **`$PYTHONPATH`** environment variable
- **`.pth`** file
- **`sys.path.insert(0, "/path")`**: please don't do that.

Now, you know how to execute softwares, congratulations !

Now, you know how to execute softwares, congratulations !

In case you have any issue at this point:

- 1 ensure you know what you try to run, with which dependencies, from where
- 2 double check if your environment variables match your expectations
- 3 try to reproduce in a pristine environment (VM / container)

Part 2: Knowledge in build systems

- Make
- Autotools
- Bazel
- CMake
- Meson

- Make
- Autotools
- Bazel
- CMake
- Meson

- setuptools
- poetry
- cargo
- npm
- yarn

```
cmake_minimum_required(VERSION 3.16)  
project(my-project LANGUAGES CXX)  
add_executable(my-bin main.cpp)
```

```
gsaurel@linux ~/ex$ cmake -B build -S .  
-- The CXX compiler identification is GNU 13.2.1  
-- Detecting CXX compiler ABI info  
-- Detecting CXX compiler ABI info - done  
-- Check for working CXX compiler: /usr/bin/g++  
-- Detecting CXX compile features  
-- Detecting CXX compile features - done  
-- Configuring done (0.3s)  
-- Generating done (0.0s)  
-- Build files have been written to:  
    /home/gsaurel/ex/build
```

```
gsaurel@linux ~/ex$ cmake --build build
[ 50%] Building CXX object CMakeFiles/my-bin.dir/main
[100%] Linking CXX executable my-bin
[100%] Built target my-bin
```

```
gsaurel@linux ~/ex$ ./build/my-bin
hello
```

```
cmake_minimum_required(VERSION 3.16)  
project(my-project LANGUAGES CXX)  
add_executable(my-bin main.cpp)  
install(TARGETS my-bin)
```

```
cmake_minimum_required(VERSION 3.16)
project(my-project LANGUAGES CXX)
add_executable(my-bin main.cpp)
install(TARGETS my-bin)
```

```
gsaurel@linux ~/ex$ cmake --build build -t install
[100%] Built target my-bin
Install the project...
-- Installing: /usr/local/bin/my-bin
CMake Error at cmake_install.cmake:52 (file):
  file INSTALL cannot copy
  file "/home/gsaurel/ex/build/my-bin"
  to "/usr/local/bin/my-bin":
  Permission denied.
```

```
gsaurel@linux ~/ex$ cmake -B build \  
    -DCMAKE_INSTALL_PREFIX=~/.my-pfx  
-- Configuring done (0.0s)  
-- Generating done (0.0s)  
-- Build files have been written to:  
    /home/gsaurel/ex/build  
gsaurel@linux ~/ex$ cmake --build build -t install  
[100%] Built target my-bin  
Install the project...  
-- Installing: /home/gsaurel/.my-pfx/bin/my-bin
```

Header: `include/my-lib.hpp`

```
#pragma once  
int add(int a, int b);
```

Source: `src/my-lib.cpp`

```
#include "my-lib.hpp"  
int add(int a, int b) { return a + b; }
```



```
add_library(my-lib SHARED
    include/my-lib.hpp
    src/my-lib.cpp)
target_include_directories(my-lib PUBLIC
    $<BUILD_INTERFACE:${CMAKE_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>)
target_link_libraries(my-bin my-lib)
install(FILES include/my-lib.hpp
    DESTINATION ${CMAKE_INSTALL_INCLUDEDIR})
install(TARGETS my-lib)
```

```
gsaurel@linux ~/ex$ cmake --build build -t install
[ 50%] Built target my-lib
[100%] Built target my-bin
Install the project...
-- Installing: /home/gsaurel/my-pfx/bin/my-bin
-- Set non-toolchain portion of runtime path
   of "/home/gsaurel/my-pfx/bin/my-bin" to ""
-- Installing: /home/gsaurel/my-pfx/include/include
-- Installing: /home/gsaurel/my-pfx/include/include/my
-- Installing: /home/gsaurel/my-pfx/lib/libmy-lib.so
```

```
gsaurel@linux ~/ex$ export PFX=/home/gsaurel/my-pfx
gsaurel@linux ~/ex$ export LD_LIBRARY_PATH=$PFX/lib
gsaurel@linux ~/ex$ export PATH=$PFX/bin
gsaurel@linux ~/ex$ my-bin
hello, 1+1 = 2
```

```
find_package(cpr REQUIRED CONFIG)  
target_link_libraries(my-lib PUBLIC cpr::cpr)
```

```
find_package(cpr REQUIRED CONFIG)  
target_link_libraries(my-lib PUBLIC cpr::cpr)
```

Configuration now requires:

```
/usr/lib/cmake/cpr/cprConfig.cmake  
/usr/lib/cmake/cpr/cprConfigVersion.cmake  
/usr/lib/cmake/cpr/cprTargets-release.cmake  
/usr/lib/cmake/cpr/cprTargets.cmake
```

```
#include "my-lib.hpp"

int main() {
    if (add(1, 1) == 3) {
        return 0;
    } else {
        return 1;
    }
}
```

```
include(CTest)
if(BUILD_TESTING)
    add_executable(my-test test.cpp)
    target_link_libraries(my-test PUBLIC my-lib)
    add_test(my-unit-test my-test)
endif()
```

```
gsaurel@linux ~/ex$ cmake --build build -t test
Running tests...
Test project /home/gsaurel/ex/build
1/1 Test #1: my-unit-test1/1 Test #1:
    my-unit-test.....***Failed      0.00 sec
```

0% tests passed, 1 tests failed out of 1

Total Test time (real) = 0.01 sec

The following tests FAILED:

1 - my-unit-test (Failed)

Errors while running CTest

gmake: *** [Makefile:71 : test] Erreur 8

- Build systems generate the compilation commands you need, and run those when necessary.
- They can be extended to take care of many other build steps, like:
 - run unit tests
 - install the package
 - build the documentation

- Build systems generate the compilation commands you need, and run those when necessary.
- They can be extended to take care of many other build steps, like:
 - run unit tests
 - install the package
 - build the documentation
- Test your unit tests: ensure they fail when they should

Part 3: Advices on package management

```

/
├── boot    # Static files of the boot loader
├── dev     # Device Files
├── etc     # System configuration
├── home
├── opt     # Add-on application packages
├── root
├── tmp
└── usr     # Second major section of the FS

```

```

/
└─ usr    # second major section of the FS
    ├── bin      # Most user commands
    ├── include  # Standard include files
    ├── lib      # Libs for packages
    │   └─ cmake
    ├── local    # Local hierarchy
    ├── share    # Architecture-independent data
    │   └─ doc
    ├── sbin     # Standard system binaries
    └─ src       # Source code
  
```

- apt
- rpm
- pacman
- apk
- ...

- apt
- rpm
- pacman
- apk
- ...

Use those and those alone to manage your OS (**sudo**, **/usr**):
anything else will have high risks of:

- security issues
- break your OS

- pip
- npm
- cargo
- cabal
- gem
- ...

- pip
- npm
- cargo
- cabal
- gem
- ...

Use those for development of your projects: they have the best support for it.

language-specific third-party package managers:

- poetry
- yarn
- stack

general package managers

- robotpkg
- catkin
- colcon
- conda
- guix
- nix

language-specific third-party package managers:

- poetry
- yarn
- stack

general package managers

- robotpkg
- catkin
- colcon
- conda
- guix
- nix

Use those if you know what you do