Packaging

UPSSITECH - KUPR9BC3

Guilhem Saurel

2025-09-26

INTRODUCTION

Goals

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

OS Scope

1. linux
2. macos, *BSD
3. ~~windows~~

OS Scope

1. linux
2. macos, *BSD
3. windows
   > with CMake / uv
   > with WSL

PART 1: UNDERSTANDING EXECUTION

Executable file

```
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
```

Environment

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

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

Binary file: source

```cpp
#include <iostream>

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

Binary file: binary

```
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
```

Shared libraries

```
gsaurel@linux ~/ex$ ldd my-bin
  linux-vdso.so.1 (0x00007fffb5ff1000)
  libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6
(0x0000722d90800000)
  libc.so.6 => /usr/lib/x86_64-linux-gnu/libc.so.6
(0x0000722d90400000)
  libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x0000722d90ab3000)
  /lib64/ld-linux-x86-64.so.2 (0x0000722d90bce000)
  libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1
(0x0000722d90a91000)
```

Shared libraries: source

```cpp
#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;
}
```

Shared libraries: binary

```
gsaurel@linux ~/ex$ g++ -o my-bin main.cpp
/usr/bin/ld : /tmp/ccAcavy5.o : dans la fonction « cpr::Response cpr::Get<cpr::Url&>(cpr::Url&) » :
main.cpp:
(.text._ZN3cpr3GetIJRNS_3UrlEEEENS_8ResponseEDpOT_[_ZN3cpr3GetIJRNS_3UrlEEEENS_8ResponseEDpOT_]+0x34) :
référence indéfinie vers « cpr::Session::Session() »
/usr/bin/ld : main.cpp:
(.text._ZN3cpr3GetIJRNS_3UrlEEEENS_8ResponseEDpOT_[_ZN3cpr3GetIJRNS_3UrlEEEENS_8ResponseEDpOT_]+0x71) :
référence indéfinie vers « cpr::Session::Get() »
/usr/bin/ld : /tmp/ccAcavy5.o : dans la fonction « void cpr::priv::set_option_internal<false,
cpr::Url&>(cpr::Session&, cpr::Url&) » :
main.cpp:
(.text._ZN3cpr4priv19set_option_internalILb0ERNS_3UrlEEEvRNS_7SessionEOT0_[_ZN3cpr4priv19set_option_internalILb0
référence indéfinie vers « cpr::Session::SetOption(cpr::Url const&) »
/usr/bin/ld : /tmp/ccAcavy5.o : dans la fonction « std::pair<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const, cpr::EncodedAuthentication>::~pair() » :
main.cpp:
(.text._ZNSt4pairIKNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEN3cpr21EncodedAuthenticationEED2Ev[_ZNSt4
référence indéfinie vers « cpr::EncodedAuthentication::~EncodedAuthentication() »
collect2: erreur: ld a retourné le statut de sortie 1
```

Shared libraries: linked binary

```
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
[…]
```

Build-time dependencies

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

Run-time dependencies

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

Run-time inspection

```
gsaurel@linux ~/ex$ ldd my-bin
  linux-vdso.so.1 (0x00007fff9188b000)
  libcpr.so.1 => /usr/lib/libcpr.so.1 (0x00007217ebc3d000)
  libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x00007217eb800000)
[…]
```

Python module

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.

Part 1: conclusion

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

Build systems

- Make
- Autotools
- Bazel
- CMake
- Meson

- setuptools
- poetry
- cargo
- npm
- yarn

`CMakeLists.txt`

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

Configure

```
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
```

Compile and run

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

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

Installation

```
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
```

Prefix

```
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
```

Library

> 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; }
```

Library: CMakeLists

```
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)
```

Library: install

```
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-lib.hpp
-- Installing: /home/gsaurel/my-pfx/lib/libmy-lib.so
```

Run

```
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
```

Dependency

```
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
```

Unit tests: source

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

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

Unit tests: CMake

```
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()
```

Unit tests: run

```
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
```

Part 2: conclusion

- 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

FHS: Filesystem Hierarchy Standard

```
/
├── 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

```
/
└── 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
```

Distribution package manager

> 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

Langage-specific official package managers

- pip
- npm
- cargo
- cabal
- gem
- …

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

Other package managers

langage-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