Modern CMake Cheat Sheet

This cheat sheet covers modern CMake features for version 3.26.3.

General Syntax

- CMake commands are in the form `COMMAND(argument1 argument2 ...)`
- Lines starting with `#` are comments
- Variables are defined with the `set` command, in the form `set(VAR_NAME value)`

Project Setup

- Begin with a minimum CMake version check:

```
cmake_minimum_required(VERSION 3.26.3)
```

- Define project name and version:

```
project(MyProject VERSION 1.0)
```

Configuring and Building Targets

- Create a target:

```
add_executable(target_name source1.cpp source2.cpp)
```

- Link a target with a library:

```
target_link_libraries(target_name PUBLIC library_name)
```

- Set properties for a target:

```
set_target_properties(target_name PROPERTIES CXX_STANDARD 17)
```

- Set include directories for a target:

```
target_include_directories(target_name PUBLIC include_dir)
```

Configuring Libraries

- Create a library:

```
add_library(library_name source1.cpp source2.cpp)
```

- Set library type:

```
set_target_properties(library_name PROPERTIES LINKER_LANGUAGE CXX)
```

- Set C compiler and version

```
set(CMAKE_C_COMPILER gcc)
set(CMAKE_C_STANDARD 11)
```

- Set library version:

```
set_target_properties(library_name PROPERTIES VERSION 1.2.3)
```

Configuring Tests

- Define a test:

```
add_test(NAME test_name COMMAND target_name)
```

- Set test timeout:

```
set_tests_properties(test_name PROPERTIES TIMEOUT 5)
```

Configuring Install

- Set installation directory:

```
install(TARGETS target_name DESTINATION bin)
```

- Install files:

```
install(FILES file1 DESTINATION share/myproject)
```

- Install directories:

```
install(DIRECTORY include/ DESTINATION include)
```

Advanced Topics

- Set compiler flags:

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -pedantic")

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
```

- Find packages:

```
find_package(PackageName REQUIRED)
```

- Conditionals:

```
if(CONDITION)
...
elseif(CONDITION2)
...
else()
...
endif()
```

Built-In Functions

`project`

This command should be called at the beginning of your CMakeLists.txt file. It sets up the project and specifies the programming language (in this case, C) and the minimum required version of CMake.

`add_compile_definitions`

Adds preprocessor definitions to a target.

```
add_compile_definitions(target_name DEFINITION1 DEFINITION2 ...)
```

`add_compile_options`

To specify C flags, you can use the add_compile_options command. For example, to add the -O2 optimization flag, you can do the following:

```
add_compile_options(-O2)
# You can add multiple flags by separating them with spaces:

add_compile_options(-Wall -Wextra -Wpedantic)
```

`add_custom_command`

Adds a custom command to a target. This command is run when the target is built.

```
add_custom_command(TARGET target_name
    POST_BUILD
    COMMAND command1 arg1 arg2
    COMMAND command2 arg1 arg2
    ...
)
```

`add_definitions`

This command allows you to define preprocessor macros. For example, to define the macro DEBUG, you can do the following:

```
add_definitions(-DDEBUG)
```

`configure_file`

Copies a file from the source directory to the build directory and replaces variables in the file with their values.

```
configure_file(input_file output_file)
```

`file`

Provides various operations for files and directories.

```
# Copy a file from source to destination
file(COPY source_file DESTINATION dest_dir)

# Copy a directory recursively
file(COPY source_dir DESTINATION dest_dir)

# Get a list of files in a directory
file(GLOB files "${CMAKE_CURRENT_SOURCE_DIR}/*.cpp")
```

`foreach`

Loops over a list of values.

```
foreach(var IN LISTS list)
    # Do something with var
endforeach()
```

`list`

Provides various operations for lists.

```
# Get the length of a list
list(LENGTH list var)

# Get a sublist
list(SUBLIST list start end)

# Append elements to a list
list(APPEND list element1 element2 ...)
```

`message`

Prints a message to the console.

```
message(STATUS "Hello, world!")
```

`set`

Sets a variable

```
set(variable value)
```

`string`

Provides various operations for strings.

```
# Get the length of a string
string(LENGTH str var)

# Get a substring
string(SUBSTRING str start length var)

# Replace all occurrences of a string
string(REPLACE old new str var)
```

Built-In Variables

`CMAKE_BINARY_DIR`

The root directory where the project is being built.

`CMAKE_CURRENT_BINARY_DIR`

The directory where the currently processed CMakeLists.txt file is located.

`CMAKE_CURRENT_LIST_FILE`

The full path to the currently processed CMakeLists.txt file.

`CMAKE_CURRENT_LIST_LINE`

The line number in the currently processed CMakeLists.txt file.

`CMAKE_CURRENT_SOURCE_DIR`

The directory where the currently processed CMakeLists.txt file is located, relative to the root source directory.

`CMAKE_SOURCE_DIR`

The root source directory of the project.

`CMAKE_SYSTEM_NAME`

The name of the current operating system.

`CMAKE_VERSION`

The version of CMake being used.

Example

```
cmake_minimum_required(VERSION 3.26)

# Set project name and version
project(MyProject VERSION 1.0)

# Set C and C++ standard
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 11)

# Add include directories
include_directories(include)

# Add subdirectories
add_subdirectory(src)
add_subdirectory(test)

# Add static library
add_library(MyStaticLib STATIC src/my_static_lib.c)
install(TARGETS MyStaticLib
    ARCHIVE DESTINATION lib
    LIBRARY DESTINATION lib
    RUNTIME DESTINATION bin)

# Add dynamic library
add_library(MyDynamicLib SHARED src/my_dynamic_lib.c)
install(TARGETS MyDynamicLib
    ARCHIVE DESTINATION lib
    LIBRARY DESTINATION lib
    RUNTIME DESTINATION bin)

# Add executable targets
add_executable(MyExecutable src/my_executable.c)
target_link_libraries(MyExecutable MyStaticLib MyDynamicLib)

add_executable(MyOtherExecutable src/my_other_executable.c)
target_link_libraries(MyOtherExecutable MyStaticLib MyDynamicLib)

# Add Google Test framework
add_subdirectory(lib/gtest)
include_directories(${gtest_SOURCE_DIR}/include ${gtest_SOURCE_DIR})

# Add unit test targets
add_executable(MyTest test/my_test.cpp)
target_link_libraries(MyTest MyStaticLib MyDynamicLib gtest gtest_main)
add_test(NAME MyTest COMMAND MyTest)

# Find external packages
find_package(CURL REQUIRED)
```

```
include_directories(${CURL_INCLUDE_DIRS})
target_link_libraries(MyExecutable ${CURL_LIBRARIES})

find_package(Boost REQUIRED COMPONENTS system filesystem)
include_directories(${Boost_INCLUDE_DIRS})
target_link_libraries(MyExecutable ${Boost_LIBRARIES})

find_package(SDL2 REQUIRED)
include_directories(${SDL2_INCLUDE_DIRS})
target_link_libraries(MyExecutable ${SDL2_LIBRARIES})

# Using if statements
# Find external packages
if(WIN32)
  # Windows-specific code
  find_package(CURL REQUIRED)
  include_directories(${CURL_INCLUDE_DIRS})
  target_link_libraries(MyExecutable ${CURL_LIBRARIES})
elseif(APPLE)
  # macOS-specific code
  find_package(Boost REQUIRED COMPONENTS system filesystem)
  include_directories(${Boost_INCLUDE_DIRS})
  target_link_libraries(MyExecutable ${Boost_LIBRARIES})
elseif(UNIX)
  # Linux-specific code
  find_package(SDL2 REQUIRED)
  include_directories(${SDL2_INCLUDE_DIRS})
  target_link_libraries(MyExecutable ${SDL2_LIBRARIES})
endif()

# Install targets

# Add packaging
set(CPACK_PACKAGE_NAME "MyProject")
set(CPACK_PACKAGE_VENDOR "MyCompany")
set(CPACK_PACKAGE_DESCRIPTION "My awesome project")
set(CPACK_PACKAGE_VERSION_MAJOR ${MyProject_VERSION_MAJOR})
set(CPACK_PACKAGE_VERSION_MINOR ${MyProject_VERSION_MINOR})
set(CPACK_PACKAGE_VERSION_PATCH ${MyProject_VERSION_PATCH})
set(CPACK_GENERATOR "ZIP")
include(CPack)
```