Makefile

January 18, 2026

1 Using Makefile to compile, build, and run, and test C++ Programs

- Makefile can automate the mundane task of compiling, recompiling, testing, and deploying of C/C++ programs
- See Makefile.pre.in file at `https://github.com/python/cpython` to see how complicated Makefile can be for large software base such as Python programming language written in C
- there's a great tutorial on Makefile: `https://makefiletutorial.com/`
- we'll demostrate a very simplified Makefile for beginners in this Notebook
- there are 5 simple demos provided in `makefile_demos` folder
- must install `make` program on your system to use Makefile
  - on Linux, `make` is usually pre-installed otherwise install it via package manager
    - on Ubuntu/Debian based systems, run `sudo apt install build-essential` command
  - on MacOS, install Xcode Command Line Tools by running `xcode-select --install` command on Terminal
  - on Windows, install `make` via Chocolatey package manager by running `choco install make` command from an elevated Command Prompt or PowerShell

1.1 Makefile Structure

- A Makefile consists of rules with the following structure

```
# comments
VARIABLE1 = value1
VARIABLE2 = value2
target: dependencies
	command1 $(VARIABLE1)
	command2 $(VARIABLE2)
	...
```

- `target` : is usually the name of the file that is generated by a program; e.g. executable or object file
- `dependencies` : are files that are used as input to create the target; e.g. source code files
- `commands` : are shell commands used to create the target from the dependencies
- each command must be preceded by a tab character (not spaces)
- when you run `make target` command, `make` will check if the target file exists and is up to date with respect to its dependencies

    - if the target file does not exist or is older than any of its dependencies, `make` will execute the commands to create or update the target file
    - if the target file is up to date, `make` will do nothing

1.2 Using Make program

- create a file named `Makefile` inside the project folder
- use Makefile template provided in makefile_demos/Makefile_template
- run the following commands from inside the project folder on a Terminal

```
$ cd projectFolder # change current working director - folder with c++ file(s)
$ make # build program
$ ls # see the name of your executable in the current directory
$ ./programName # run the program by it's name
$ make clean # run clean rule; usually deletes all object/exe files
```

1.2.1 Note

- You typically run make commands from Terminal
- for demonstration, we'll use Jupyter Notebook to run the make commands
    - Jupyter Notebook can run Bash commands with `!` symbol
    - Ipython Kernel is required to run the magic commands that starts with `%` on Jupyter notebook

[1]:
```
! pwd # print the current working directory
```

```
/Users/rbasnet/projects/CPP-Fundamentals/notebooks
```

[6]:
```
%cd demos/makefiles
```

```
/Users/rbasnet/projects/CPP-Fundamentals/notebooks/demos/makefiles
```

[7]:
```
! ls
```

```
Makefile_template demo2             demo4
demo1             demo3
demo5
```

[8]:
```
%cd demo1
```

```
/Users/rbasnet/projects/CPP-Fundamentals/notebooks/demos/makefiles/demo1
```

[9]:
```
! ls
```

```
Makefile  hello.cpp
```

1.3 Makefile Demos

- `demos/makefiles/` folder contains 5 simple Makefile demos

1.3.1 demo1

- has three simple rules
- compile, run and clean
- rule names end with :
  - you can have one or more commands associated with the rule
  - commands are tab indented
  - commands are tyically Bash commands that you normally run on Terminal
- rules are called from Terminal using syntax:

```
make <rule_name>
```

- e.g.,

```
make run
```

- while running make, if no rule name is called, the first rule is executed by default

[10]:
```
! cat Makefile
```

```
# a simple Makefile with 3 rules

# rule for compiling program
# make or make compile triggers the following rule
compile:
        g++ hello.cpp

# rule for running programming
# make run triggers the following rule
run:
        ./a.out

# rule for clean up
# make clean triggers the following rule
clean:
        rm -f a.out
```

[7]:
```
! make # run make compile, the first rule by default
```

```
g++ hello.cpp
```

[8]:
```
! ls
```

```
Makefile  a.out     hello.cpp
```

[9]:
```
! make run
```

```
./a.out
Hello World!
```

[12]:
```
%cd ..
```

```
/Users/rbasnet/projects/CPP-Fundamentals/notebooks/demos/makefiles
```

[13]:
```
%cd demo5
```

```
/Users/rbasnet/projects/CPP-Fundamentals/notebooks/demos/makefiles/demo5
```

1.3.2 demo5

- run Makefile from terminal
- for some reason it doesn't exectue from Jupyter Notebook

[14]:
```
! ls
```

```
Makefile  hello.cpp
```

[13]:
```
! cat Makefile
```

```
# Farily complex Makefile demo

COMPILER = clang++
COMPILER_FLAGS = -c -g -Wall -std=c++17

# list .cpp files separated by space
CPP_FILES = hello.cpp

# executable program name
PROGRAM_NAME = hello.exe

# rule using other rules
# other rules must be written after the rule name on the same line separated by
a space
all: build run clean
        @echo "All Done!"


# rule for compiling and building program
# make or make all or make build triggers the following rule
# @ suppreses/hides the command itself from printing
build:
        @# compile .cpp to object file .o
        @echo "compiling…"
        $(COMPILER) $(COMPILER_FLAGS) $(CPP_FILES)
        @# build executable from object files
        @echo "building…"
        $(COMPILER) -o $(PROGRAM_NAME) *.o

# rule for running binary program
# make run triggers the following rule
run:
        @echo "running program…"
        ./$(PROGRAM_NAME)
```

```
# rule for clean up
# make clean triggers the following rule
clean:
	@echo "cleaning up…"
	@rm -f $(PROGRAM_NAME) *.o *.out 2> /dev/null
```

[16]:
```
! make build
```

```
compiling…
clang++ -c -g -Wall -std=c++17 hello.cpp
building…
clang++ -o hello.exe *.o
```

[17]:
```
! make run
```

```
running program…
./hello.exe
Hello World!
```

[18]:
```
! ls
```

```
Makefile  hello.cpp hello.exe hello.o
```

[19]:
```
! make clean
```

```
cleaning up…
```

[20]:
```
! ls
```

```
Makefile  hello.cpp
```

[ ]:
```

```