Системы сборки

Николай Пономарев

27 октября 2025 г.

Санкт-Петербургский
государственный университет

Системы сборки

- Среда разработки не всегда доступна
  - Сборочные сервера автоматически выполняют сборку после каждого коммита, там некому открыть IDE и нажать на кнопку «запустить»
- Воспроизводимость сборки
  - Если чтобы собрать программу надо открыть проект, скопировать пару десятков файлов, поправить кое-какие пути и делать это в полнолуние, то возможны ошибки
- Автоматизация сборки
  - git clone
  - одна консольная команда, которая всё делает за нас
  - ...
  - готовое к работе приложение

Сборка вручную без IDE

```
$ gcc main.c
```

или, например,

```
$ gcc  -Wall -Wextra -pedantic -O2 -g -c lib1
$ gcc  -Wall -Wextra -pedantic -O2 -g -c lib2
$ gcc  -Wall -Wextra -pedantic -O2 -g -c lib3
$ gcc  -Wall -Wextra -pedantic -O2 -g -lm lib1.o lib2.o lib3.o main.c ...
```

- Если проект большой, это быстро становится грустно
- Десятки тысяч файлов — не редкость

make

- Стандарт де-факто по «низкоуровневым» правилам сборки
- Сама ничего не знает про языки программирования, компиляторы и прочие подобные штуки
- Знает про цели и зависимости целей
  - Сортирует граф целей
  - Цели применяются в порядке от листьев к корню
- Правила сборки описываются в Makefile

Жаждущие подробностей могут посмотреть на слайды в конце презентации

Пример

Синтаксис:

```
target [target ...]: [component ...]
    [command 1]
    .
    .
    .
    [command n]
```

Пример:

```
hello:
    gcc -Wall -Wextra -pedantic -O2 hello.c -o hello
```

Запуск:

```
$ make hello
```

Высокоуровневые системы сборки

- Либо сами вызывают необходимые инструменты, либо генерируют Makefile
- CMake
  - Кроссплатформенная система сборки, очень популярна в C и C++ open source-сообществе
- Великое множество других (Bazel, MSBuild, qmake, SCons, Meson, ...)

Написание скриптов сборки для большого проекта — отдельная и довольно трудоёмкая задача

CMake

- Декларативное описание сборки проекта
  - Указываем *что* собирать, а не *как* собирать
- С открытым исходным кодом
  - https://gitlab.kitware.com/cmake/cmake
- Прежде всего для сборки C++-проектов, но умеет много чего
- Разрабатывается с 1999 года
- Сама сборкой не занимается, генерирует конфиг для системы сборки
  - Но может сама их запускать
- Пожалуй, основной способ сборки в VS Code (плагином CMake Tools)
  - Ctrl+Shift+P → CMake: Quick Start

Конфигурация CMake

- CMakeLists.txt в корне проекта
- Также могут быть в подкаталогах (для иерархичной конфигурации)
- Бывают также модули (.cmake) и скрипты
- Коммитить, соответственно, CMakeLists.txt (все) и .cmake, если есть

Основные понятия

- Команды
  - `command_name(список аргументов через пробел)`
  - В CMake всё команды
  - Всё строки
- Переменные
  - `set(name value)`
  - `message(STATUS "Name = ${name}")`
  - Переменные окружения: `$ENV{Имя}`
- Функции

Цели

- Исполняемые файлы
  - `add_executable(targetname source1 ...)`
- Библиотеки
  - `add_library(targetname [STATIC | SHARED | ...] source1 ...)`
- Линковка целей
  - `target_link_libraries(myLib [PUBLIC | PRIVATE | ...] dependencyLib)`

Фазы сборки

- Конфигурация
  - `$ cmake <путь до папки с CMakeLists.txt>`
  - Читает `CMakeLists.txt`, строит модель, сохраняет в `CMakeCache.txt`
  - Пытается угадать наличествующие инструменты и их возможности (например, поддерживаемую компилятором версию стандарта)
  - Можно вручную подредактировать параметры конфигурации (руками или в `cmake-gui`)
- Генерация — генерация конфигов для сборки (например, Makefile)
- Сборка
  - `$ cmake --build`

Пример

```
# Устанавливаем минимальную версию CMake
cmake_minimum_required(VERSION 3.25)
# Указываем названием проекта и используемый язык(и)
project(example C)

# Перечисляем библиотеки
add_library(Input Input.c)
add_library(Logic Logic.c)
add_library(Output Output.c)
# Связывает Logic с Output
target_link_libraries(Logic PUBLIC Output)

# Указываем исполняемый файл
add_executable(main main.c)
# Связываемся с библиотеками, Output доступен из-за Logic
target_link_libraries(main PRIVATE Input)
target_link_libraries(main PRIVATE Logic)
```

```
.
├── CMakeLists.txt
├── Input.c
├── Input.h
├── Logic.c
├── Logic.h
├── main.c
├── Output.c
└── Output.h
```

```
$ cmake . -B build
$ cmake --build build
$ ./build/main
```

Out-of-source-сборка

```
source/
├── build
│   └── сгенерированные файлы
├── CMakeLists.txt
└── исходники
```

```
build/
└── сгенерированные файлы
source/
├── CMakeLists.txt
└── исходники
```

Что ещё умеет

- Ветвления
  - Обычно на основе предопределённых переменных: https://cmake.org/cmake/help/latest/manual/cmake-variables.7.html
- Конфигурации: Debug, Release, RelWithDebInfo, MinSizeRel
  - `$ cmake ../MyProject -DCMAKE_BUILD_TYPE=Debug`
- Подпапки/подпроекты
  - `add_subdirectory(sourceDir ...)`
- Toolchain-файлы
- Скачивать исходники прямо в процессе сборки!
  - FetchContent
  - См., например, https://google.github.io/googletest/quickstart-cmake.html

Домашнее задание

Обязательное задание (4 балла):
Перенести на CMake домашнее задание к 5 занятию (Продвинутый баланс скобок и Сортировочная станция)

Дополнительные задания (не входят в общий зачёт):
Перенести остальные домашние работы на CMake (1 балл за каждую домашку)

Ещё слайды про make

Далее несколько слайдов про make, которые не влезли в пару

Как работает make

make знает про цели, зависимости, временные штампы и правила

- Смотрит на зависимости цели, если у хоть одной временной штамп свежее цели, запускается правило для цели
- В процессе цель может обновить свой временной штамп, что приведёт к исполнению правил для зависящих от неё целей
- Цели и зависимости образуют направленный ациклический граф (DAG)
- make выполняет топологическую сортировку графа зависимостей
- Правила применяются в порядке от листьев к корню

Продвинутые штуки

- Переменные
  - *MACRO* = definition
  - *NEW_MACRO* = $(MACRO) - $(MACRO2)
  - Переопределение из командной строки
    - $ make *MACRO*=ololo
- Параллельная сборка
  - $ make -j8

Неявные правила (Implicit rules)

- Способ описания общих правил
- Пример:

```
%.txt : %.docx
    pandoc $< -o $@
```

  Использование: `$ make test.txt`, при существовании `test.docx`
- Реализованы для множества языков (C, C++, Pascal, Fortran, ...)
- Можно собрать однофайловую программу на Си: `$ make main`
- Или даже собрать небольшой проект из нескольких файлов:

```
CFLAGS += -Wall -Wextra -pedantic -O2 -g
main : lib1.o lib2.o lib3.o
```

Подробнее: https://www.gnu.org/software/make/manual/make.html#Implicit-Rules