

# Консоль и системы сборки

Юрий Литвинов  
y.litvinov@spbu.ru

17.11.2023

# Консоль, зачем

- ▶ Программы с интерфейсом командной строки
  - ▶ В Linux-подобных системах очень многие программы имеют только интерфейс командной строки
- ▶ Пакетный режим и автоматизация
- ▶ Удалённое управление
- ▶ Не везде есть графический интерфейс

# Интерпретаторы командной строки, терминалы

## Командные интерпретаторы

- ▶ Windows
  - ▶ cmd
  - ▶ Windows Power Shell
- ▶ Linux
  - ▶ bash, zsh, csh, ...

## Терминалы

- ▶ Windows — cmd, far, cmder
- ▶ Linux — terminal, xterm, konsole, yaquake, ...
- ▶ Удалённое управление — PuTTY, ssh

# cmd

- ▶ Есть в любой Windows из коробки, но лучше поставить Far
  - ▶ Win-R, Cmd
- ▶ Команды
  - ▶ dir, cd, xcopy, mkdir, del, ...
  - ▶ ключ /?
- ▶ Пути
  - ▶ ./ololo.exe = ololo.exe
  - ▶ ../ololo.exe
  - ▶ относительный путь: ../myProgram/bin/ololo.exe
  - ▶ абсолютный путь: C:/myProgram/bin/ololo.exe
- ▶ Поток — stdout, stderr
  - ▶ echo "test" > someFile.txt 2> errors.txt

## .bat-файлы

- ▶ `echo "Hello, world"`
- ▶ Параметры командной строки
  - ▶ `%1, %2, ..., %*`
- ▶ Циклы, условия, `goto` и т.д.
  - ▶ `for /l %x in (1, 1, 100) do echo %x`
- ▶ `rem` Это комментарий
- ▶ `@echo off`
- ▶ `call` — вызов другого скрипта
- ▶ `cmd /C` — создание нового командного интерпретатора

`Ctrl-C` — прервать выполнение скрипта (если что-то пошло не так),  
`Ctrl-D` — конец входного потока

## .sh-файлы

- ▶ Зависят от интерпретатора (bash, zsh, csh, ...)
  - ▶ `#!/bin/bash` — “shebang”
- ▶ `echo “Hello, world”`
- ▶ Параметры командной строки
  - ▶ `$0, $1, ..., $#, $@`
- ▶ Циклы, условия и т.д.
  - ▶ 

```
if ! [ -f "ololo.txt" ]; then
    echo "File not found"
    exit 1
fi
```
  - ▶ 

```
for i in $@; do
    echo $i
done
```
- ▶ `chmod +x ./test.sh`

# Код возврата

cmd

- ▶ Можно проверить тот самый return -1 из main-а и в зависимости от него сделать или не сделать что-нибудь

test.exe

```
if not errorlevel 0 (  
    echo Everything is bad  
) else (  
    echo Everything is good  
)
```

# Переменные окружения (Windows)

- ▶ %<имя переменной>%
- ▶ echo %path%
- ▶ set OLOLO=ololo
- ▶ Глобальный контекст
  - ▶ “Панель управления” -> “Система” -> “Дополнительные параметры системы” -> “Переменные среды”
  - ▶ setx — требует админских прав
- ▶ PATH

У каждого процесса свой контекст

- ▶ Working Directory
- ▶ Своя копия переменных окружения на момент запуска
- ▶ Контекст наследуется от процесса-родителя



# Переменные окружения (Linux)

- ▶ `$<имя переменной>`
- ▶ `echo $PATH`
- ▶ `export OLOLO=ololo`
- ▶ Глобальный контекст
  - ▶ `~/.bashrc` — скрипт, исполняющийся при старте командного интерпретатора
  - ▶ Туда можно писать что угодно
    - ▶ И сломать себе всё
- ▶ `PATH`

# Системы сборки

- ▶ Среда разработки не всегда доступна
  - ▶ Continuous Integration-сервера автоматически выполняют сборку после каждого коммита, там некому открыть Visual Studio 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

- ▶ Visual Studio:

`cl <имя .c-файла>`

или, например,

`cl /W4 /EHsc file1.c file2.c file3.c /link /out:program1.exe`

- ▶ Работает только из Developer Command Prompt (ну, почти только)

- ▶ g++

`g++ <имя .c-файла>`

или, например,

`g++ -Wall -o helloworld helloworld.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, 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
- ▶ Правила сборки описываются в Makefile

# Пример

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

Пример:

```
hello: ; @echo "hello"
```

# Продвинутые штуки

- ▶ Переменные
  - ▶ MACRO = definition
  - ▶ NEW\_MACRO = \$(MACRO) - \$(MACRO2)
  - ▶ Переопределение из командной строки
    - ▶ make MACRO=ololo
- ▶ Суффиксные правила

`.SUFFIXES: .txt .html`  
*# From .html to .txt*  
`.html.txt:`  
`lynx -dump $< > $@`
- ▶ Параллельная сборка
  - ▶ make -j8

## Под Windows

- ▶ mingw32-make
  - ▶ Используется в mingw (“Minimalist GNU for Windows”)
    - ▶ Порт gcc на Windows
- ▶ nmake
  - ▶ Реализация от Microsoft, в комплекте с Visual Studio
    - ▶ Запускается из Developer Command Prompt

Мейкфайлы зависят от конкретной реализации make (Makefile от mingw32-make может не собраться nmake-ом)

# Высокоуровневые системы сборки

- ▶ Либо сами вызывают необходимые инструменты, либо генерируют мейкфайлы
- ▶ MSBuild
  - ▶ Собирает из консоли .sln, .vcxproj, .csproj и т.д. -файлы
  - ▶ Тоже запускается из Developer Command Prompt
- ▶ CMake
  - ▶ Кроссплатформенная система сборки, очень популярна в C- и C++- open source-сообществе

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



# CMake

- ▶ С открытым исходным кодом  
(<https://gitlab.kitware.com/cmake/cmake>)
- ▶ Прежде всего для сборки C++-проектов, но умеет много чего
- ▶ Разрабатывается с 1999 года
- ▶ Бывает версии 2 и 3
- ▶ 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

# Минимальный пример

```
cmake_minimum_required(VERSION 3.20)  
project>Hello)
```

```
add_executable>Hello Hello.c)
```

# Out-of-source-сборка

source

build

сгенерированные файлы

CMakeLists.txt

исходники

build

сгенерированные файлы

source

CMakeLists.txt

исходники

# 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, `.vsxproj`-файла)
- ▶ Сборка
  - ▶ `cmake --build`

# 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

## Ветвления

```
if (UNIX)
    message(STATUS "Running on Unix")
else()
    message(STATUS "Running on not Unix")
endif()
```

- ▶ Предопределённые переменные: <https://cmake.org/cmake/help/latest/manual/cmake-variables.7.html>
- ▶ Всё это работает на этапе конфигурации!

# Цели

- ▶ Декларативное описание того, что хочется собрать
- ▶ Исполняемые файлы
  - ▶ `add_executable(targetname source1 ...)`
- ▶ Библиотеки
  - ▶ `add_library(targetname [STATIC SHARED | ...] source1 ...)|`
- ▶ Линковка целей
  - ▶ `target_link_libraries(myLib [PUBLIC PRIVATE | INTERFACE] dependencyLib)|`



# Что ещё умеет

- ▶ Конфигурации: Debug, Release, RelWithDebInfo, MinSizeRel
  - ▶ `cmake ../MyProject -DCMAKE_BUILD_TYPE=Debug`
- ▶ Подпапки/подпроекты
  - ▶ `add_subdirectory(sourceDir ...)`
- ▶ Toolchain-файлы
- ▶ Укачивать исходники прямо в процессе сборки!
  - ▶ См., например,  
<https://google.github.io/googletest/quickstart-cmake.html>

# Continuous Integration

- ▶ Эталонное и единое для проекта окружение, в котором выполняется сборка
  - ▶ Сборка выполняется очень часто, иногда — после каждого коммита
- ▶ Там же запускаются юнит-тесты
- ▶ Сборка запускается в свежей виртуальной машине или Docker-контейнере
- ▶ Управляется конфигурацией, хранящейся прямо в репозитории

# GitHub Actions

- ▶ Бесплатная система облачной сборки для проектов на GitHub
- ▶ <https://docs.github.com/en/actions>
- ▶ Как настроить:
  - ▶ В репозитории на GitHub Settings -> Actions -> Allow all actions
  - ▶ Создаём в корне репозитория папку .github/workflows/
  - ▶ В нём создаём файл <имя действия>.yml (например, ci.yml)
  - ▶ Описываем процесс сборки согласно <https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions>
    - ▶ Пример и описание линуксовой сборки: <https://www.incredibuild.com/blog/using-github-actions-with-your-c-project>
    - ▶ Пример .NET-сборки: <https://github.com/yurii-litvinov/DocUtils/blob/master/.github/workflows/ci.yml>
- ▶ Коммитим-пушим
- ▶ Смотрим статус коммита и пуллреквеста