

Консоль и системы сборки

Юрий Литвинов
yurii.litvinov@gmail.com

09.11.2018

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, cmdr
- ▶ 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
- ▶ Потоки — 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

- ▶ Сильно 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`

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 .cpp-файла>`

или, например,

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

- ▶ gcc

`g++ <имя .cpp-файла>`

или, например,

`g++ -Wall -o helloworld helloworld.cpp`

- ▶ 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
 - ▶ Можно поставить отдельно, можно в составе Qt SDK
- ▶ nmake
 - ▶ Реализация от Microsoft, в комплекте с Visual Studio
 - ▶ Запускается из Developer Command Prompt
 - ▶ На самом деле, это просто консоль с правильно выставленными переменными окружения

Мейкфайлы зависят от операционной системы и даже 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++ open source-сообществе
- ▶ qmake
 - ▶ Qt-specific

Написание скриптов сборки для большого проекта – отдельная и трудоёмкая задача

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
- ▶ <https://travis-ci.org/> — пример бесплатной для open source-проектов облачной CI-системы
 - ▶ Интегрируется с GitHub-ом
 - ▶ Собирает на чистой виртуальной машине под Ubuntu 12.04 (или Ubuntu 14.04) или OS X
 - ▶ Окружение и правила сборки настраиваются конфигурационным файлом, который должен лежать в корне репозитория