Cmake – кроссплатформная система сборки

Аннотация

Cmake представляет собой переносимую системы подготовки исходных кодой для компиляции. На основе единого определения исходных кодов можно организовать сборку исполняемой программы для одной или множества платформ. В статье описано применение Cmake, а также использование этой системы в KDE для создания новых программ.

Кроссплатформная система Cmake

Часто ли вам приходилось сталкиваться с ситуациями, когда для загруженного из сети дерева исходных кодов того или иного приложения используемая в вашей системе версия autotools отказывалась работать? Система autotools включает широкий набор средств подготовки, но зачастую они оказываются несовместимыми и компилируемым кодом и для правки autotools требуется понимать работу таких средств, как make, m4, сценарии командного процессора. Более того, система autotools разрослась до такой степени, что потребовались дополнительные инструменты типа automake. Тем не менее во многих проектах СПО autotools продолжают использовать.

Однако некоторые разработчики переключились на использование других инструментов типа Cross Platform Make или CMake, которые добавили ряд отсутствующих в autotools возможностей. Подобно autotools, система CMake позволяет генерировать файлы Unix Makefile. Однако, в отличие от autotools, CMake может также создавать файлы KDevelop, Visual Studio, XCode (Apple) на основе одного конфигурационного файла. Более того, команды настройки конфигурации и сборки собраны в одном месте и полностью обслуживаются CMake. Благодаря отмеченному и ряду других преимуществ перед autotools, система CMake была выбрана в качестве стандарта для настройки и сборки программных пакетов в KDE4.

Многие дистрибутивы Linux включают CMake, однако версии пакета могут оказаться достаточно старыми. В последнее время в CMake было добавлено множество новых функций и следует использовать программу с номером версии не ниже 2.4.x. Загрузить программу можно с сайта разработчика http://www.cmake.org/HTML/Download.html. Для сборки KDE4 требуется CMake версии не ниже 2.4.1. В конце мая 2013 года была выпущена версия 2.8.11.

После загрузки свежего пакета с сайта распакуйте архив в тот или иной каталог на диске. CMake использует сомое себя для сборки, поэтому сначала нужно из каталога с деревом исходных кодов выполнить команду bootstrap, после чего воспользоваться командой make для компиляции и make install для установки Cmake.

```
$ ./bootstrap
$ make
$ sudo make install
```

если не было задано никаких опций, CMake будет устанавливаться в каталог /usr/local/. Если вы хотите поместить пакет в иное место, задайте в команде bootstrap параметр вида —prefix=/имяКаталога/.

Использование CMake

После установки CMake с программой можно сразу же начинать работу. Для этого потребуется проект, использующий CMake. Поскольку сам пакет CMake относится к числу таких проектов, можно поупражняться с ним. Создайте новый каталог, распакуйте туда архив исходных кодов и перейдите в этот каталог.

Одной из наиболее важных концепций CMake является подготовка дерева для сборки в отдельном каталоге. В общем случае CMake позволяет собирать пакет в каталоге дерева исходных кодов, но на практике делать этого не рекомендуется. При сборке программы в отдельном каталоге общее дерево исходных кодов может служить для сборки различных вариантов пакет (например, рабочего и отладочного), а также при сборке пакета для разных платформ (просто вы создает в этом случае отдельный каталог для каждой сборки).

После создания каталога для сборки перейдите в него и введите команду:

```
$ cmake /путь_к_дереву_исходных_кодов
```

По этой команде будет выполнена настройка конфигурационных файлов CMake и генерация файлов Unix Makefile, которые могут далее использоваться при обычной компиляции с помощью команды make. Если вы используете интегрированную среду разработки (IDE) типа KDevelop, программа CMake позволит работать и с ней. Для генерации проектных файлов KDevelop просто введите команду:

```
$ cmake -G KDevelop3 /путь_к_дереву_исходных_кодов
```

В результате будет выполнена настройка конфигурационных файлов CMAKE и созданы проектные файлы Kdevelop, которые можно будет использовать для компиляции.

Если вам не нравится работа из командной строки, можно воспользоваться curses-интерфейсом Cmake, который позволяет задать опции сборки в интерактивном режиме. Этот интерфейс вызывается по команде ccmake и работает подобно CMake. Просто введите из каталога сборки команду:

```
$ ccmake /путь_к_дереву_исходных_кодов
```

Эта команда активизирует интерфейс curses для всех конфигурационных переменных пакета. Если вы воспользуетесь ccmake до CMake, интерфейс программы окажется практически пустым. Не пугайтесь этого - CMake агрегирует

конфигурационные опции из всех файлов дерева исходных кодов собираемого пакета в общий кэш-файл после первой настройки конфигурации.

Итак, перейдем к настройке конфигурации с помощью интерфейса ccmake. Нажмите клавишу **c** для настройки. В строке состояния появятся те или иные сообщения, а вскоре после этого на экран будут выведены те или иные переменные. Если переменных на экране не появилось, нажмите клавишу **t** для перехода в расширенный режим, где доступны все переменные конфигурации. Для перемещения по списку переменных служат клавиши управления курсором (стрелки вверх и вниз. Для изменения значения переменной, на которой расположен указатель, нажмите клавишу Enter.

Одной из наиболее часто изменяемых переменных является CMAKE_INSTALL_PREFIX. Как можно видеть из названия переменной, ее значение определяет базовый каталог для установки собранного пакета. По умолчанию эта переменная обычно имеет значение **/usr/local**, но вы может задать другой каталог. Поместите указатель на переменную CMAKE_INSTALL_PREFIX, нажмите клавишу Enter и введите новое имя каталога (например, /opt/CMAKE) или отредактируйте имеющееся. После ввода нужного имени нажмите клавишу Enter для записи нового значения. После этого нажмите клавишу **c** для настройки конфигурации и клавишу **g** для генерации файлов Makefile. Теперь после компиляции программы она будет установлена в каталог /opt/CMAKE после ввода команды make install.

Префикс каталога для установки пакета можно задать из командной строки с помощью простой команды:

```
$ cmake –DCMAKE_INSTALL_PREFIX=/opt/CMAKE
```

которая установит для CMAKE_INSTALL_PREFIX значение /opt/CMAKE.

Создание проектов CMake

Рассмотрим создание проекта на основе Cmake.

Команды CMake сохраняются в файле CMakeLists.txt. Ниже приведен файл CMakeLists.txt для простого примера программы «Hello, World!» на языке C:

```
PROJECT(hello C)
ADD_EXECUTABLE(hello hello.c)
```

Первая строка файла указывает проект на языке C по имени hello. Вторая строка задает сборку программы hello из исходного файла hello.c. Если исходный код содержится в нескольких файлах, эти файлы просто перечисляются вслед за hello.c. Сейчас можно собрать проект с помощью тех же команд, которые применялись для сборки CMake.

Усложним проект, добавив в него библиотеку, код которой содержится в специальном файле:

```
PROJECT (hello C)
ADD_LIBRARY(hellolib STATIC hellolib.c)
ADD_EXECUTABLE(hello hello.c)
TARGET_LINK_LIBRARIES(hello hellolib)
```

Здесь создается статически компонуемая библиотека hellolib из файла hellolib.c и компонуется в программу hello. В команде TARGET_LINK_LIBRARIES можно задавать также системные библиотеки.

Если бы возможности CMake исчерпывались перечисленными выше, программа не давала бы преимуществ по сравнению с обычными файлами Makefile. Однако CMake обеспечивает полный контроль над опциями, которые проект может использовать. В качестве примера рассмотрим проект, который может включать компоненты для взаимодействия с USB и позволяет в момент компиляции выбрать включение этой компоненты или отказ от нее. Файл CMakeLists.txt для такого проекта будет иметь вид:

```
PROJECT(myproject)
OPTION(WITH_USB "Include our USB component" OFF)
SET(SRCS file1.c file2.c file3.c)
IF(WITH_USB)
  SET(SRCS ${SRCS} usbfile1.c usbfile2.c)
ENDIF(WITH_USB)
ADD_EXECUTABLE(myproject ${SRCS})
```

Строка OPTION(...) создает параметр WITH_USB и устанавливает для него по умолчанию значение OFF. Строка включает также справочную информацию (help string), описывающее назначение данной опции. С учетом значения WITH_USB, в значение переменной SRCS (список файлов с исходным кодом) включаются или не включаются два файла, связанных с USB - usbfile1.c и usbfile2.c. Полученная в результате переменная SRCS передается при вызове ADD_EXECUTABLE для задания списка файлов с исходным кодом проекта. Для включения поддержки USB следует просто включить опцию WITH_USB в команде:

```
$ cmake –DWITH_USB=ON /путь_к_дереву_исходных_кодов
```

Пойдем дальше и рассмотрим проект, где необязательная поддержка USB реализована в форме библиотеки, исходные файлы которой размещены в отдельном субкаталоге. Новый файл CMakeLists.txt будет иметь вид:

```
PROJECT(myproject)
OPTION(WITH_USB "Include our USB component" OFF)
SET(SRCS file1.c file2.c file3.c)
ADD_EXECUTABLE(myproject ${SRCS})
IF(WITH_USB)
  ADD_DIRECTORY(usblib)
```

```
  TARGET_LINK_LIBRARIES(myproject usblib)
ENDIF(WITH_USB)
```

Для этого проекта исходный код библиотеки поддержки USB помещается в субкаталог usblib вместе со своим файлом CmakeLists.txt, имеющим вид:

```
PROJECT(usblib)
SET(SRCS usbfile1.c usbfile2.c)
ADD_LIBRARY(usblib ${SRCS})
```

После этого при включенной поддержке USB программа CMake будет собирать библиотеку USB и компоновать ее в исполняемый файл проекта.

Как сделать информацию об использовании необязательного кода USB доступной? Для начала связанный с USB код следует поместить в #ifdef WITH_USB. CMake будет обрабатывать специальный конфигурационный файл, заменяя соответствующие метки значениями переменных CMake.

Новый конфигурационный файл config.h.cmake будет иметь вид:

```
#ifndef TEST_CONFIG_H
#define TEST_CONFIG_H

#cmakedefine WITH_USB

#endif
```

А основной файл CMakeLists.txt слегка изменяется, приобретая вид:

```
PROJECT(myproject)
OPTION(WITH_USB "Include our USB component" OFF)
CONFIGURE_FILE(${CMAKE_SOURCE_DIR}/config.h.cmake
  ${CMAKE_BINARY_DIR}/config.h)
INCLUDE_DIRECTORIES(${CMAKE_SOURCE_DIR} ${CMAKE_BINARY_DIR})
```

С такими файлами CMake транслирует config.h.cmake, создавая заголовочный файл config.h в каталоге сборки проекта. Команда INCLUDE_DIRECTORIES говорит компилятору о необходимости добавления каталога с исходным кодом и каталога сборки в путь поиска включаемых файлов (include). Если опция WITH_USB включена, CMake заменит #cmakedefine WITH_USB на #define WITH_USB, а при отключенной опции — на /*#undef WITH_USB*/. Как только будет включен файл config.h и код USB помещен в #ifdef WITH_USB, все должно заработать. Единственным неудобством является необходимость вручную создавать файл config.h.cmake. Программа CMake включает стандартные макросы для проверки заголовочных файлов, функций, переменных, библиотек, символов и размера типов. Например, если вы хотите проверить наличие заголовочного файла unistd.h, следует использовать команды:

```
INCLUDE(CheckIncludeFile)
CHECK_INCLUDE_FILE(unistd.h HAVE_UNISTD)
```

Эти команды проверят наличие файла unistd.h и установят соответствующее значение переменной HAVE_UNISTD. После этого достаточно добавить #cmakedefine HAVE_UNISTD в файл config.h.cmake.

Поиск функций осуществляется путем включения CheckFunctionExists и использования макроса CHECK_FUNCTION_EXISTS. Для проверки переменных служит включение CheckVariableExists и применение макроса CHECK_VARIABLE_EXISTS. В обоих случаях синтаксис совпадает с CHECK_INCLUDE_FILE.

Для поиска библиотек служит макрос CHECK_LIBRARY_EXISTS. При его использовании требуется указать имя библиотеки, имя обеспечиваемой библиотекой функции, место поиска, а также задать переменную для записи результата поиска. Например, для проверки динамической библиотеки можно использовать команды:

```
INCLUDE(CheckLibraryExists)
CHECK_LIBRARY_EXISTS(dl dlopen "" HAVE_LIBDL)
```

Эти команды будут проверять наличие функции dlopen() в библиотеке libdl и при положительном результате поиска устанавливать для переменной HAVE_LIBDL значение true. CMake автоматически проверяет стандартные системные библиотеки, поэтому место поиска в данном случае не задано.

При помске символов отыскиваются одновременной имена символов и функций. Кроме символа и выходной переменной в команде требуется указать заголовочные файлы для включения в поиск. Ниже приведен пример поиска символа LC_MESSAGES в заголовочном файле locale.h и установки переменной HAVE_LC_MESSAGES по результату поиска:

```
INCLUDE(CheckSymbolExists)
CHECK_SYMBOL_EXISTS(LC_MESSAGES "locale.h" HAVE_LC_MESSAGES)
```

Проверка размера переменных выполняется несколько иным способом. В этом случае выходная переменная указывает размер проверяемого типа в байтах, а дополнительная переменная HAVE_ ${VARIABLE} указывает наличие или отсутствие нужного типа.

```
INCLUDE(CheckTypeSize)
CHECK_TYPE_SIZE (int INT_SIZE)
```

На 32-разрядных машинах эти команды будут устанавливать для переменной INT_SIZE значение 4 (байта), а для HAVE_INT_SIZE — значение true.

В дополнение к механизмам проверки конфигурационных опций нижнего уровня CMake включает опции поиска пакетов. Например, многие программы Linux могут использовать среду X Windows System Version 11 (X11). Для простой проверки наличия X11 и выбора опций компоновки и включения заголовочных файлов можно использовать команду:

```
FIND_PACKAGE(X11 REQUIRED)
```

Необязательный параметр REQUIRED говорит CMake о необходимости прекращения работы и возврата кода ошибки в случае отсутствия искомой программы. Команда FIND_PACKAGE (X11 REQUIRED) устанавливает ряд переменных:

- X11_FOUND = true при наличии X11;
- X11_INCLUDE_DIR указывает каталоги include для использования функций X11;
- X11_LIBRARIES указывает библиотеки, компонуемые для использования X11.

Две последних переменные могут быть переданы непосредственно в INCLUDE_DIRECTORIES и TARGET_LINK_LIBRARIES. CMake обеспечивает модули поиска для множества стандартных пакетов, узнать о которых можно с помощью команды cmake --help-module-list, выводящей полный список установленных модулей, включая модули FIND_PACKAGE.

При возникновении некритических ошибок CMake позволяет скомпилировать и запустить программы C и C++ с использованием команд TRY_COMPILE и TRY_RUN. Более подробно это описано в документации CMake.

CMake включает мощную справочную систему по поддерживаемым программой командам и модулям. Команда cmake —help позволяет увидеть базовую информацию, cmake —help-command commandname и cmake —help-module modulename дают справку по указанной команде или модулю. Для получения списков поддерживаемых программой команд и модулей служат команды cmake —help-command-list и cmake —help-module-list.

CMake и KDE4

Разобравшись с основами применения CMake, можно начать использование CMake для сборки KDE4? Для этого потребуется Cmake версии не ниже 2.4.1. Потребуеться также пакет Qt 4.1.1 или выше и svn-версии kdelibs и kdebase. (см. http://developer.kde.org/source/anonsvn.html).

Сборка KDE4 должна выполняться вне дерева исходных кодов. Поэтому нужно будет создать каталог для сборки kdelibs. Перейдя в созданный каталог введите команду:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/kde4 \
  -DCMAKE_BUILD_TYPE=debug \
  /path/to/kdelib/sources
```

Библиотеки kdelibs будут собраны с поддержкой отладочной информации и установлены в каталог /opt/kde4. Если вы хотите получить расширенную отладочную информацию, задайте опцию debugfull. KDE4 поддерживает аткже несколько дополнительных пакетов. После первоначальной настройки конфигурации Cmake вы можете воспользоваться командой ccmake из каталога сборки для настройки всех опций.

Завершив настройку конфигурационных параметров, введите команды

```
$ make
$ sudo make install
```

для сборки и установки библиотек kdelibs. После инсталяции kdelibs можно таким же путем собрать и установить пакет kdebase. Для получения дополнительной информации настоятельно рекомендуется прочесть файл COMPILING-WITH-CMAKE в дереве исходных кодов kdelibs и документы на сайте KDE Wiki. При возникновении тех или иных вопросов по поводу cmake и KDE вы можете адресовать их в почтовую конференцию kde-buildsystem.

Сборка может быть простой

Простой синтаксис Cmake позволяет разработчикам задать опции настройки конфигурации и сборки пакетов так, чтобы у конечного пользователя не возникало проблем при установке программ из исходных кодов.

Перевод на русский язык
Николай Малых
nmalykh@protocols.ru

Николай Малых

nmalykh@protocols.ru