Лекция 2. Средство построения проектов GNU make

Разработка многоплатформенного ПО

7 сентября 2016 г.

Модульный подход к разработке ПО

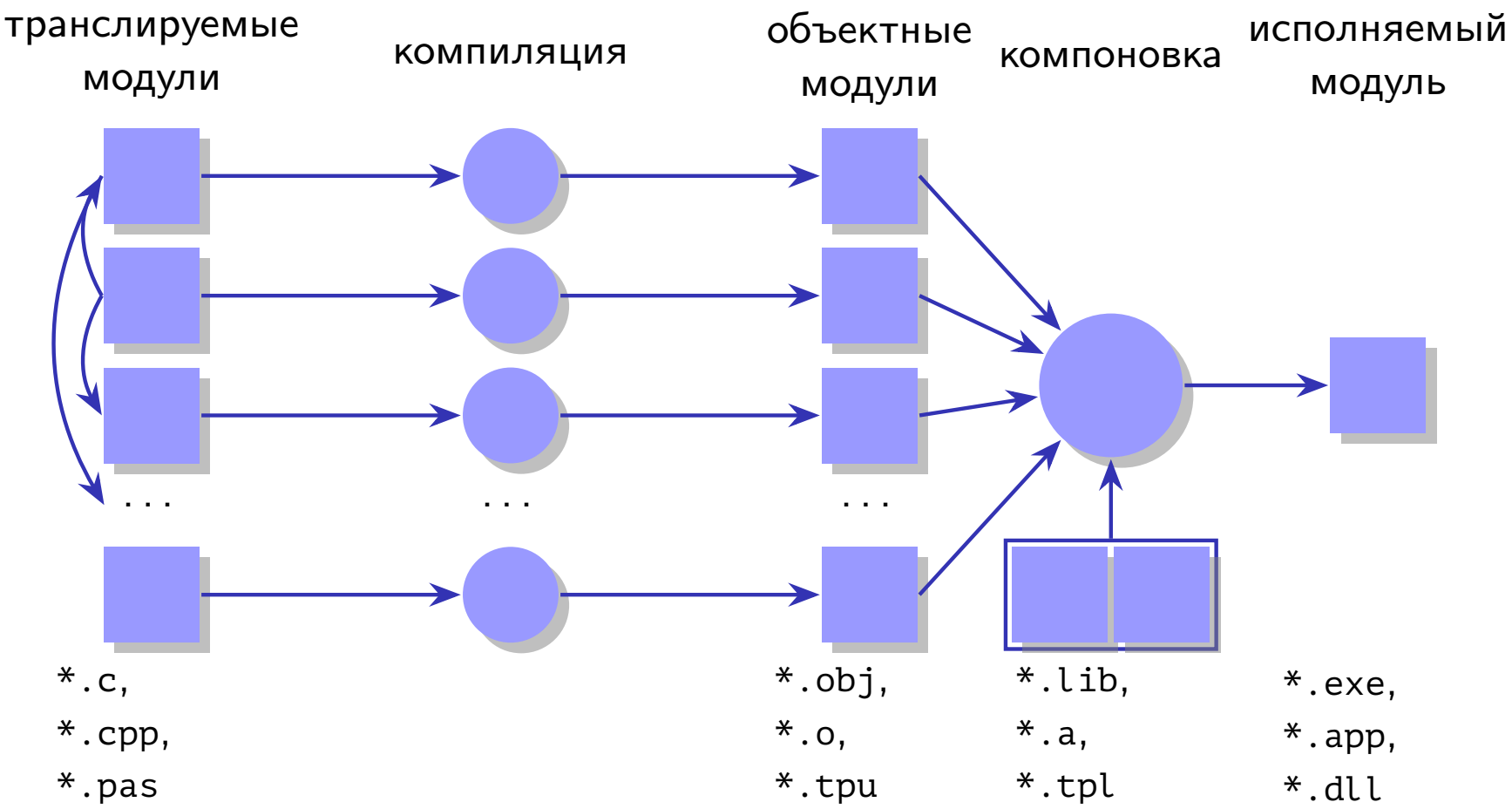

Рис. 1: процесс построения проекта с модульным подходом

Основные преимущества

Преимущества использования утилиты make

- Удобство в описании сложных проектов.
- Возможность автоматизации сборки.
- Поддержка инкрементной сборки.
- Универсальность.
- Нетребовательность к вычислительным ресурсам.

Основные реализации

Реализации утилиты make

- Unix make (1977, Stuart Feldman, AT&T Bell Labs).
- GNU make (1988, Richard Stallman, Roland McGrath, FSF) — Linux, MinGW, Cygwin.
- BSD make — FreeBSD, NetBSD, OpenBSD.
- Microsoft nmake.
- Borland make.
- Watcom wmake.
- ...

Простой проект приложения

a.cpp

```
#include <iostream>

void f()
{
  std::cout << "f() from a.cpp" << std::endl;
}
```

b.cpp

```
#include "a.h"

int main()
{
  f();
}
```

a.h

```
void f();
```

Использование драйвера компилятора

Пример (команды оболочки)

```
$ g++ -o test_make a.cpp b.cpp
$ ./test_make
f() from a.cpp
$
```

Структура проекта

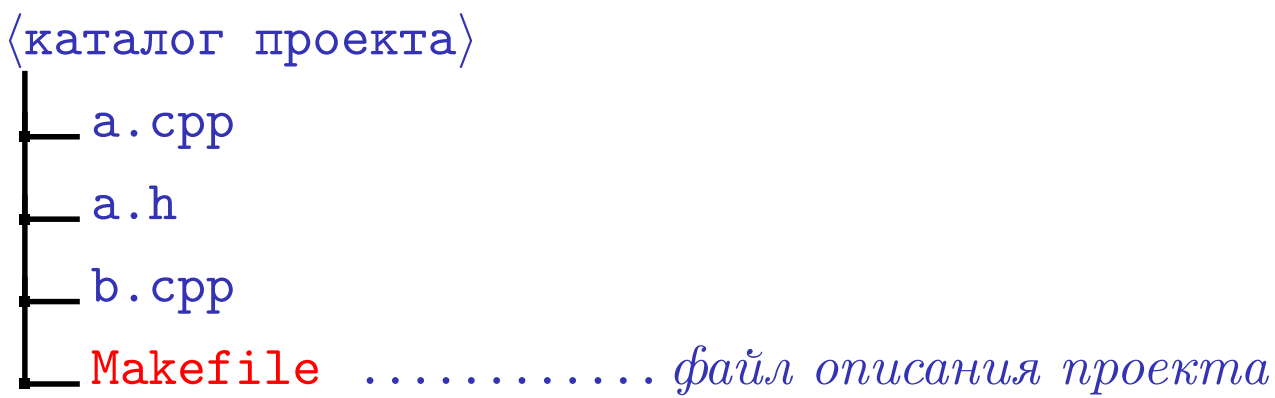

Рис. 2: структура каталога проекта

Структура make-файла

Описание цели в make-файле

 # Комментарий

⟨*цель*⟩ [⟨*цель*⟩ ...]: [⟨*предусловие*⟩ ...]
[⟨*TAB*⟩⟨*команда 1*⟩]
...
[⟨*TAB*⟩⟨*команда n*⟩]

Пример make-файла

Пример (Makefile)

```
# Makefile

all: test_make

test_make: a.o b.o
	g++ -o test_make a.o b.o

a.o: a.cpp a.h
	g++ -c -o a.o a.cpp

b.o: b.cpp a.h
	g++ -c -o b.o b.cpp
```

Пример (команды оболочки)

```
$ make
g++ -c -o a.o a.cpp
g++ -c -o b.o b.cpp
g++ -o test_make a.o b.o
$ ./test_make
f() from a.cpp
$ touch a.cpp
$ make
g++ -c -o a.o a.cpp
g++ -o test_make a.o b.o
$
```

Автоматические переменные

Переменная	Значение
$@	имя файла цели текущего правила;
$+	имена всех предусловий, разделённые пробелами;
$^	имена всех предусловий, разделённые пробелами, без повторов;
$<	имя первого предусловия;
$|	имена всех предусловий, определяющих только порядок.

Таблица 1: основные автоматические переменные

Использование автоматических переменных

Пример (Makefile)

```
test_make: a.o b.o
	g++ -o $@ $^

a.o: a.cpp a.h
	g++ -c -o $@ a.cpp

b.o: b.cpp a.h
	g++ -c -o $@ b.cpp

clean:
	rm -f *.o
	rm -f test_make
```

Пример (команды оболочки)

```
$ make clean
rm -f *.o
rm -f test_make
$ make
g++ -c -o a.o a.cpp
g++ -c -o b.o b.cpp
g++ -o test_make a.o b.o
$
```

Использование шаблонов

Пример (Makefile)

```
out/test_make: out/a.o out/b.o
        g++ -o $@ $^

out/%.o: %.cpp a.h
        mkdir -p out
        g++ -c -o $@ $<

clean:
        rm -rf out
```

Использование функций подстановки строк

Пример (Makefile)

```
SOURCES := a.cpp b.cpp

out/test_make: $(patsubst %.cpp, out/%.o, ${SOURCES})
#          или: $(SOURCES:%.cpp=out/%.o)
        g++ -o $@ $^

out/%.o: %.cpp a.h
        mkdir -p out
        g++ -c -o $@ $<

clean:
        rm -rf out
```

Операции присваивания

Операция	Действие	Замена правой части
:=	присваивание значения	в месте присваивания;
=	присваивание значения	в месте использования переменной левой части;
?=	присваивание значения, если не присвоено	в месте использования переменной левой части;
+=	конкатенация через пробел	зависит от предыдущей установки переменной;

Таблица 2: операции присваивания

Различия в присваиваниях

Пример (Makefile)

```
A   = abc
BA := $(A)def
BB  = $(A)def
A   = x

all:
        echo $(BA)
        echo $(BB)
```

Использование переменных

Пример (Makefile)

```
        SOURCES  := a.cpp b.cpp
override CXXFLAGS += -std=c++11
override LDFLAGS  += -lm

out/test_make: $(SOURCES:%.cpp=out/%.o)
	g++ ${LDFLAGS} -o $@ $^

out/%.o: %.cpp a.h
	mkdir -p out
	g++ ${CPPFLAGS} ${CXXFLAGS} -c -o $@ $<

clean:
	rm -rf out
```

Работа утилиты make

Пример (команды оболочки)

```
$ make CXXFLAGS=-O2
mkdir -p out
g++  -O2 -std=c++11 -c -o out/a.o a.cpp
mkdir -p out
g++  -O2 -std=c++11 -c -o out/b.o b.cpp
g++ -lm -o out/test_make out/a.o out/b.o
$ make clean
rm -rf out
$
```

Стандартные переменные

Переменная	Значение
CC	команда вызова компилятора C;
CXX	команда вызова компилятора C++;
AR	команда вызова архиватора для библиотек.

Таблица 3: основные переменные для имён исполняемых файлов

Переменная	Значение
CFLAGS	аргументы компилятора C;
CXXFLAGS	аргументы компилятора C++;
CPPFLAGS	аргументы препроцессора;
LDFLAGS	аргументы сборщика.

Таблица 4: основные переменные аргументов команд

Пример установки имени исполняемого файла

Пример (Makefile)

```
CC := gcc-4.3

# ...
```

Пример (команды оболочки)

```
$ make CC=gcc-4.4
...
```

Множественные цели и правила

Пример (Makefile)

```
# ...

out/%.o: %.cpp
        mkdir -p out
        ${CXX} ${CPPFLAGS} ${CXXFLAGS} -c -o $@ $<

out/a.o out/b.o: a.h     # повторные правила

# ...
```

Фальшивые цели

Пример (Makefile)

```
# ...

.PHONY: clean          # объявление фальшивой цели

clean:
        rm -rf out
```

Соглашения об именах фальшивых целей

Цель	Действия
`all`	(цель по умолчанию) скомпилировать всю программу без сборки документации и установки;
`check`	запустить самотестирование, программа должна быть предварительно собрана, но не обязательно установлена;
`clean`	удалить все файлы, сгенерированные в результате нормального выполнения сборки — нет необходимости удалять каталоги, созданные при помощи команды `mkdir -p`;
`install`	скомпилировать всю программу и скопировать все файлы в каталоги системы, где они должны находиться для реального использования;
`uninstall`	удалить все установленные файлы.

Таблица 5: основные часто используемые имена целей в GNU-программах

Предусловия, определяющие только порядок

Описание цели в make-файле

⟨*цели*⟩ : [⟨*обычные предусловия*⟩] [| ⟨*предусловия только порядка*⟩]
[⟨*рецепты*⟩]

Пример предусловия, определяющего только порядок

Пример (Makefile)

```
# ...

out/%.o: %.cpp | out
	${CXX} ${CPPFLAGS} ${CXXFLAGS} -c -o $@ $<

# ...

out:
	mkdir -p out

# ...
```

Работа утилиты make

Пример (команды оболочки)

```
$ make
mkdir -p out
g++   -c -o out/a.o a.cpp
g++   -c -o out/b.o b.cpp
g++ -lm -o out/test_make out/a.o out/b.o
$
```

Структура проекта с библиотекой

Рис. 3: структура каталога проекта

Работа утилиты make

Пример (команды оболочки)

```
$ make
mkdir -p out
g++ -c -o out/main.o main.cpp
g++ -c -o out/a.o a.cpp
g++ -c -o out/b.o b.cpp
ar cr out/libab.a out/a.o out/b.o
g++ -o out/main out/main.o out/libab.a
$
```

Пример включения внешних make-файлов

Пример (Makefile)

```
include  ../common.mak ../variables.mak
-include may_not_exist.mak

# ...
```

Пример рекурсивных make-файлов

Пример (Makefile)

```
SUBDIRS = progect1 project2 project3

.PHONY: subdirs $(SUBDIRS)

subdirs: $(SUBDIRS)

$(SUBDIRS):
	$(MAKE) -C $@

project1: project3
```

Работа утилиты make

Пример (команды оболочки)

```
$ make
make -C project3
make[1]: Entering directory ‘/home/stu003/work/cross/projects/project3’
echo project3 > /dev/null
make[1]: Leaving directory ‘/home/stu003/work/cross/projects/project3’
make -C project1
make[1]: Entering directory ‘/home/stu003/work/cross/projects/project1’
echo project1 > /dev/null
make[1]: Leaving directory ‘/home/stu003/work/cross/projects/project1’
make -C project2
make[1]: Entering directory ‘/home/stu003/work/cross/projects/project2’
echo project2 > /dev/null
make[1]: Leaving directory ‘/home/stu003/work/cross/projects/project2’
```

Структура проекта с подкаталогами

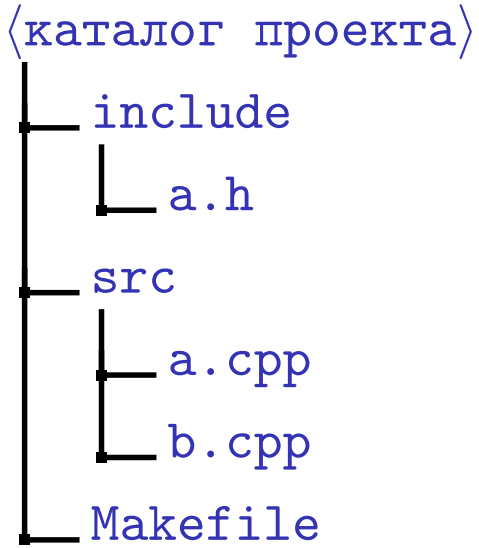

Рис. 4: структура каталога проекта

Пример использования директивы **vpath**

Пример (Makefile)

```
        SOURCES  := a.cpp b.cpp
override CPPFLAGS += -I include

vpath %.h   include
vpath %.cpp src
vpath %.o   _out

_out/test_make: $(SOURCES:%.cpp=_out/%.o) | _out
        $(CXX) $(LDFLAGS) -o $@ $^

_out/%.o: %.cpp a.h | _out
        $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c -o $@ $<
```

Пример автоматического поиска зависимостей

Пример (Makefile)

```
        SOURCES   = a.cpp b.cpp
override CPPFLAGS += -MMD -MP -I include

# ...
vpath %.d   _out

_out/hello_world: $(SOURCES:%.cpp=_out/%.o) | _out
        $(CXX) $(LDFLAGS) -o $@ $^

_out/%.o: %.cpp | _out
        $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c -o $@ $<

-include $(SOURCES:%.cpp=_out/%.d)
```