1 Утилита make

Когда разрабатываемая программа состоит более чем из одного исходного `.c` файла, ил
когда для получения нужного результата (например, исполняемой программы), нужно вы
полнить много зависящих друг от друга действий, задавать команды компиляции для каж
дого файла вручную становится очень сложно, если вообще возможно. Для работы с про
ектами сложной структуры используется утилита **make**. Утилита входит в комплект поста
ки всех **Unix**-подобных систем и во все инструментальные системы программирования д
DOS/Windows. Однако, синтаксис управляющего файла очень сильно отличается в **Unix**
не-**Unix** системах, поэтому далее будет рассматриваться только **Unix**-версия.

Описание проекта для утилиты **make** содержится в специальном файле, который мо
жет называться произвольным образом, но по традиции обычно называется `Makefile` ил
`makefile`. Если имя файла проекта явно не задано, при запуске утилита ищет в текуще
каталоге файл с указанными выше именами, и, если такой файл существует, выполняет ко
манды из него.

Утилита **make** отслеживает зависимости между компонентами проекта, и, если один и
файлов был модифицирован, все файлы, которые от него зависят, перекомпилируются.

Файл описания проекта может содержать описания переменных, описания зависимосте
и описания команд, которые используются для компиляции. Каждый элемент файла описа
ния проекта должен, как правило, располагаться на отдельной строке. Для размещения эле
мента описания проекта на нескольких строках используется символ продолжения \ точн
так же, как в директивах препроцессора языка Си.

Определения переменных записываются следующим образом:

```
<имя> = <определение>
```

Использование переменной записывается в одной из двух форм: `$(<имя>)` или `${<имя>`
Переменные рассматриваются как макросы, то есть использование переменной означа
подстановку текста из определения переменной в точку использования. Если при опреде
лении переменной были использованы другие переменные, подстановка их значений прои
ходит при использовании переменной (опять так же, как и в препроцессоре языка Си).

В правилах, в особенности в шаблонных правилах, можно использовать специальны
переменные, которые раскрываются в зависимости от правила, в котором они стоят. Пер
менная `$@` раскрывается в имя цели, стоящей в левой части правила. Переменная `$<` ра
крывается в имя первой зависимости в правой части правила. Переменная `$^` раскрывает
в список всех зависимостей в правой части.

Зависимости между компонентами определяются следующим образом:

```
<цель> : <цель1> <цель2> ... <цельn>
```

Где `<цель>` — имя цели, которое может быть либо именем файла, либо некоторым им
нем, обозначающим действие, которому не соответствует никакой файл, например `clea`
Список целей в правой части задает цели, от которых зависит `<цель>`.

Цель, стоящая в левой части правила, считается *устаревшей*, если необходимо выпо
нить перекомпиляцию для её обновления. Это происходит в одном из трёх случаев:

1. Одна из целей, стоящих в правой части правила, является устаревшей.
2. Файл с именем `<цель>` не существует.

3. Файл с именем одной из целей, стоящих в правой части правила, имеет более позднюю дату модификации, чем файл с именем <цель>.

Для всех устаревших целей выполняются команды перекомпиляции в порядке, чтобы цель, зависящая от другой цели, будет компилироваться после цели, от которой она зависит. Утилита **make** имеет набор стандартных правил, например, для получения `.o` из `.c` файлов. Кроме того, команды может явно указывать пользователь. Для этого после правила, определяющего зависимость записывается одна или несколько команд. Команды должны идти с отступом в один символ табуляции от начала строки. Пробелы в качестве отступа не допускаются.

Существуют так называемые шаблонные правила. Такие правила определяют не конкретные зависимости одних файлов от других, а команды, которые выполняются, если зависимости удовлетворяют некоторому шаблону. В этом случае команды в зависимостях можно опускать. Например, шаблонное правило для зависимостей .o файлов от .c файлов может выглядеть следующим образом:

```
.c.o:
	$(CC) -c $(CFLAGS) $<
```

Утилита **make** имеет набор встроенных шаблонных правил. Описанное выше правило является встроенным.

Пример файла описания проекта `makefile` приведен ниже.

```
# Makefile for project calculator
TARGET=calc
CC=gcc
CFLAGS=-g -Wall
OBJECTS=calc.o mylib.o

#don't change this
all:$(TARGET)
clean:
	rm -f *.o $(TARGET) core
$(TARGET): $(OBJECTS)
	$(CC) -o $(TARGET) $(CFLAGS) $(OBJECTS)
.c.o:
	$(CC) -c $(CFLAGS) $<
#dependency from h files
calc.o: mylib.h
```

Комментарии записываются после символа # и до конца строки. Зависимости `.o` файлов от `.c` файлов с тем же корнем можно явно не указывать, так как утилита **make** «знает» о таких зависимостях.

Обычно переменной `CC` соответствует имя компилятора, `TARGET` название исполняемого файла-результата компиляции, `OBJECTS` список объектных файлов проекта.

Для того чтобы настроить приведенный выше пример на другой проект, нужно поменять список файлов в `OBJECTS`, список зависимостей `.o` и `.h` файлов, и название исполняемого файла `TARGET`.

Если файл описания проекта называется `makefile` или `Makefile`, утилита `make` запускается с помощью команды `make [ <цели> ]`. Если файл описания проекта называется по-другому, нужно еще указать опцию `-f <имя скрипта>`. Если цель в командной строке не указана, выполняется первая цель в описании проекта. То есть, для компиляции достаточно выполнить команду `make`. Для очистки рабочего каталога можно выполнить команду `make clean`.

Значение переменных можно менять из командной строки. Например, для компиляции с использованием компилятора `cc`, а не `gcc`, можно выполнить команду `make CC=cc`.

Когда проект становится достаточно большим, поддерживать правильный список зависимостей `.c` от `.h` файлов становиться достаточно затруднительно. Чтобы получить список зависимостей можно воспользоваться специальной возможностью компилятора **gcc**. Указание ключа `-MM` в командной строке **gcc** указывает, что на стандартный поток вывода должны быть напечатаны правила зависимостей `.c` файлов от подключаемых файлов. Фрагмент файла описания проекта может выглядеть следующим образом:

```
HFILES=a.h b.h c.h d.h # список всех .h файлов
CFILES=a.c b.c c.c d.c # список всех .c файлов
SRCFILES=$(CFILES) $(HFILES)

deps.make: $(SRCFILES)
	gcc -MM $(CFILES) > deps.make
include deps.make
```