Отчет по лабораторной работе №13

Дисциплина: Операционные системы

Шишук Владислав Олегович

Содержание

# Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# Выполнение лабораторной работы

1. В домашнем каталоге создаем подкаталог ~/work/os/lab\_prog (рис.1)

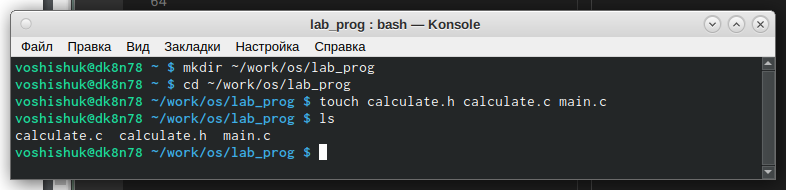


рис.1

1. Создаем в нем файлы calculate.h, calculate.c, main.c.(рис.1) Далее реализуем функции калькулятора в файлах: calculate.c(рис.2,рис.3),calculate.h(рис.4),main.c(рис.5)

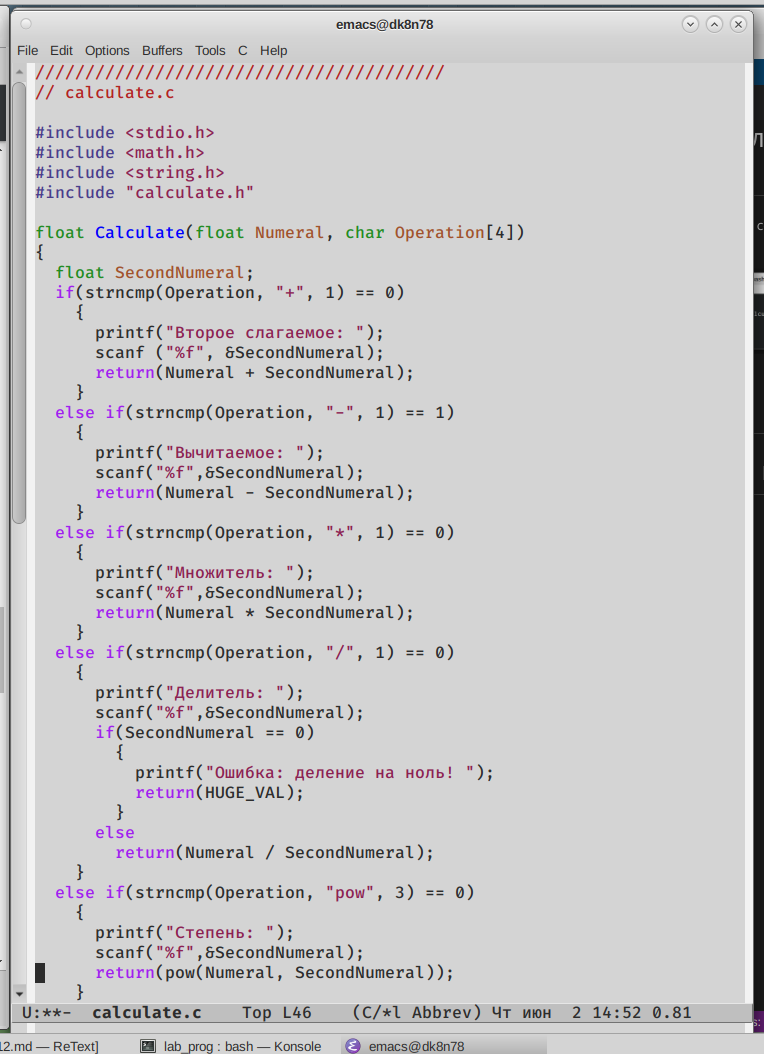


рис.2

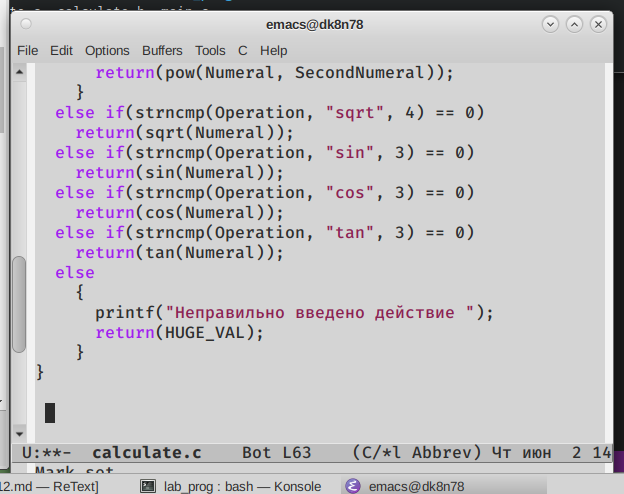


рис.3

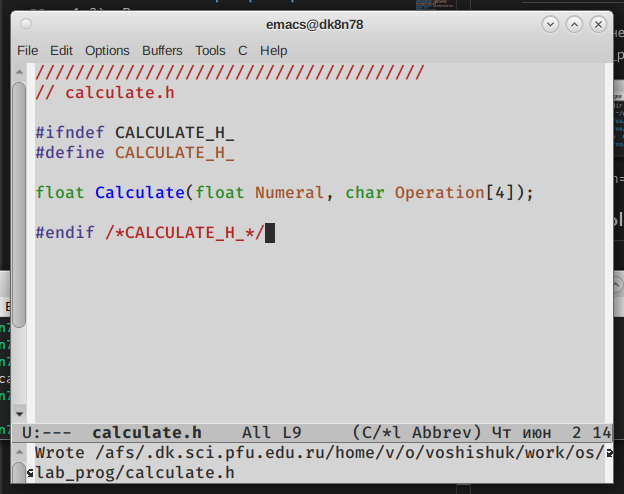


рис.4



рис.5

1. Выполнили компиляцию посредством gcc(рис.6):

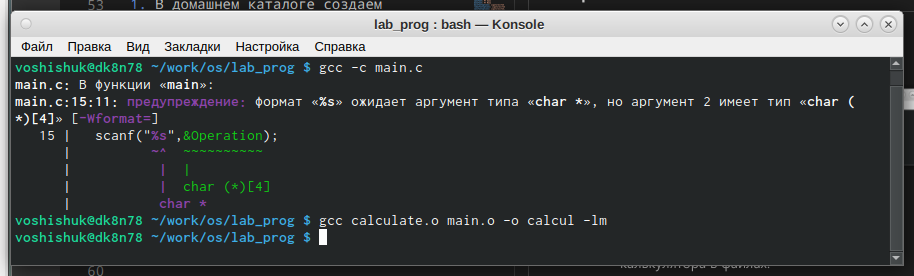


рис.6

1. Ошибок не возникло.
2. Создаем Makefile с нужным содержанием. Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл. (рис.7)

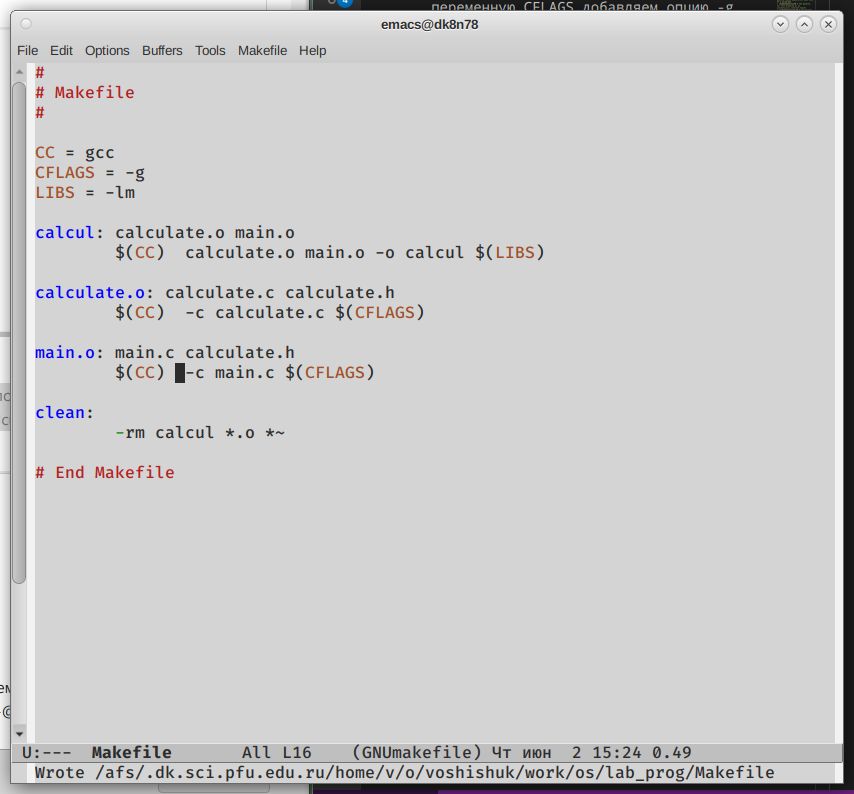


рис.7

1. Makefile уже исправлен: в переменную CFLAGS добавляем опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB; lелаем так, что утилита компиляции выбирается с помощью переменной CC.  
   Выполняем компиляцию файлов, используя команды «make calculate.o», «make main.o», «male calcul».(рис.8)

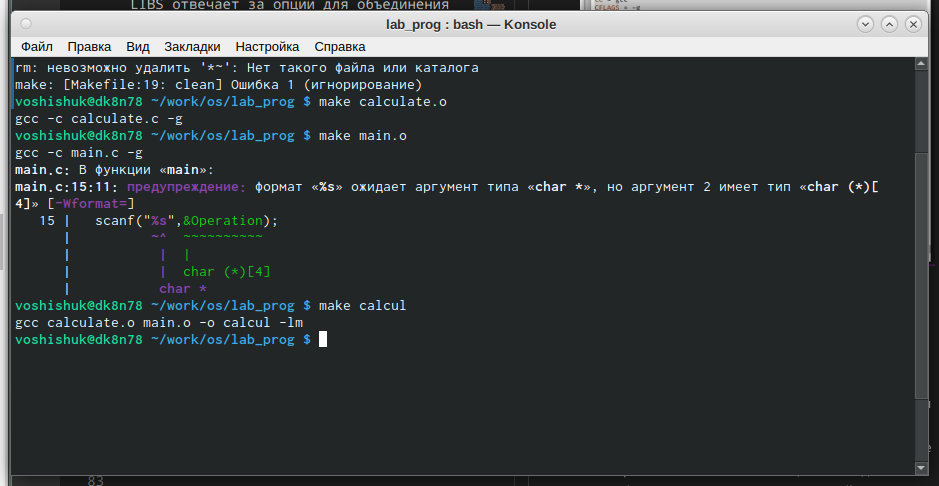
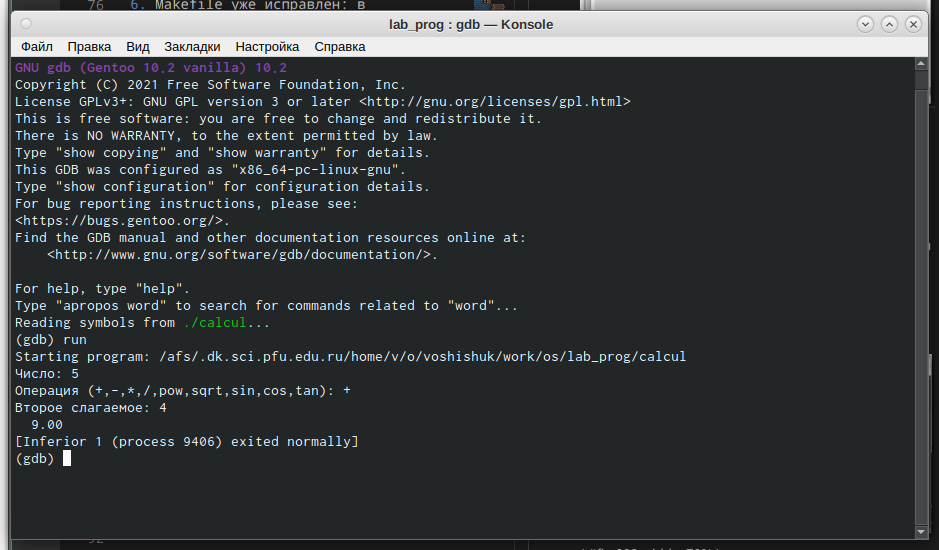


рис.8

* Запустили отладчик GDB, загрузив в него программу для отладки
* Для запуска программы внутри отладчика ввели команду run(рис.9)

 - Для постраничного (по 9 строк) просмотра исходного код использовали команду list - Для просмотра строк с 12 по 15 основного файла использовали list с параметрами(рис.10)

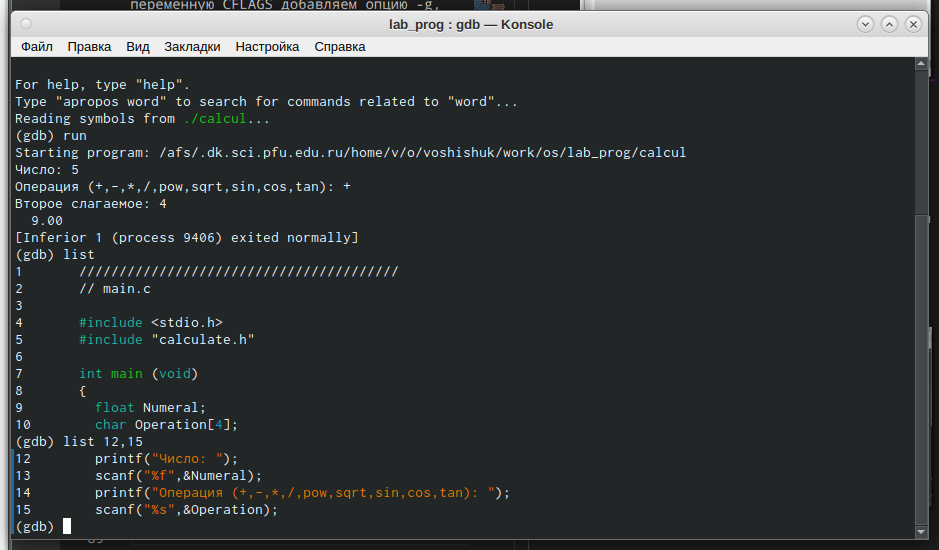


рис.10

* Для просмотра определённых строк не основного файла использовали list с параметрами
* Установили точку останова в файле calculate.c на строке номер 21(рис.11)

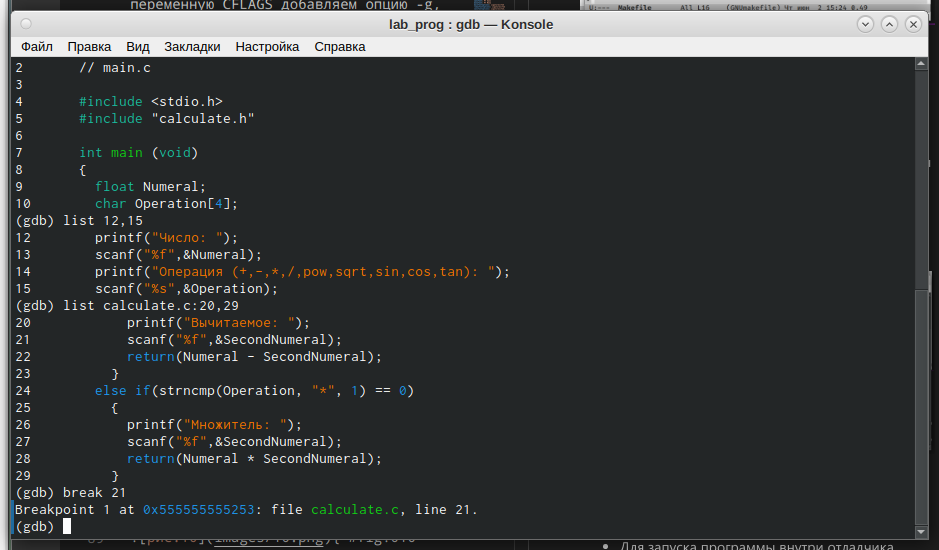


рис.11

* Вывели информацию об имеющихся в проекте точка останова
* Запустили программу внутри отладчика и убедились, что программа остановится в момент прохождения точки останова:

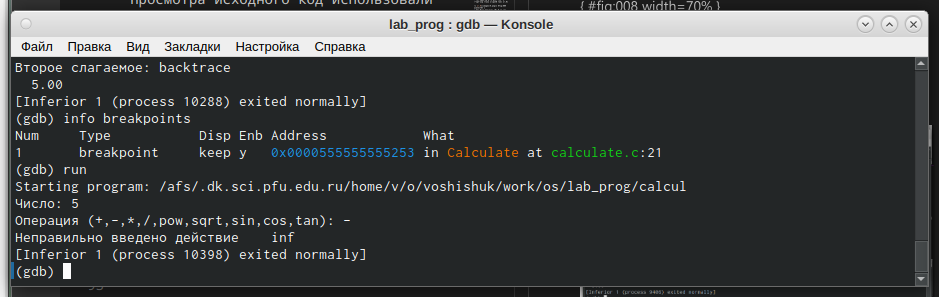


рис.12

* Убрали точку останова(рис.13)

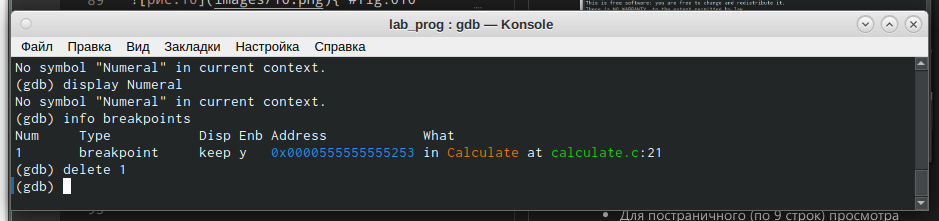


рис.13

1. С помощью утилиты splint анализируем коды файлов calculate.c и main.c. Предварительно устанавливаем данную утилиту с помощью команды «yum install splint». Далее используем команду «splint calculate.c» и «splint main.c». C помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохранятся. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потери данных.(рис.14-15)

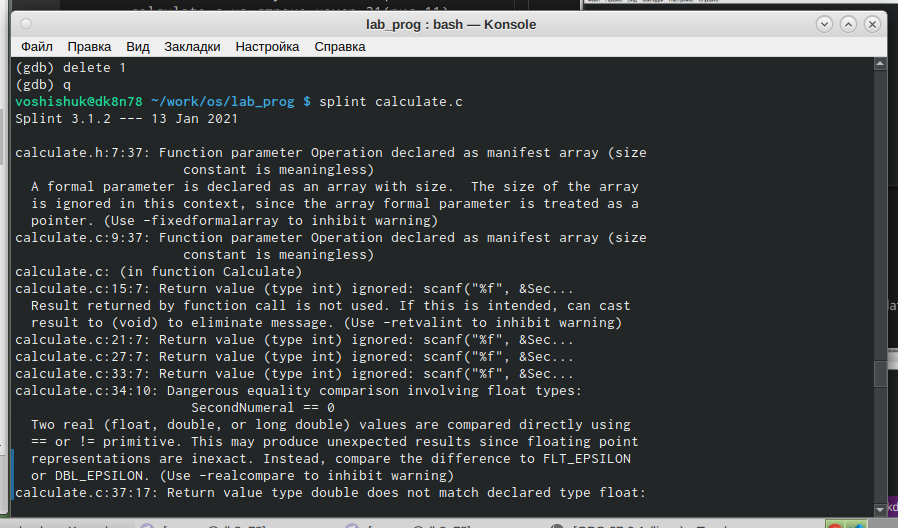


рис.14

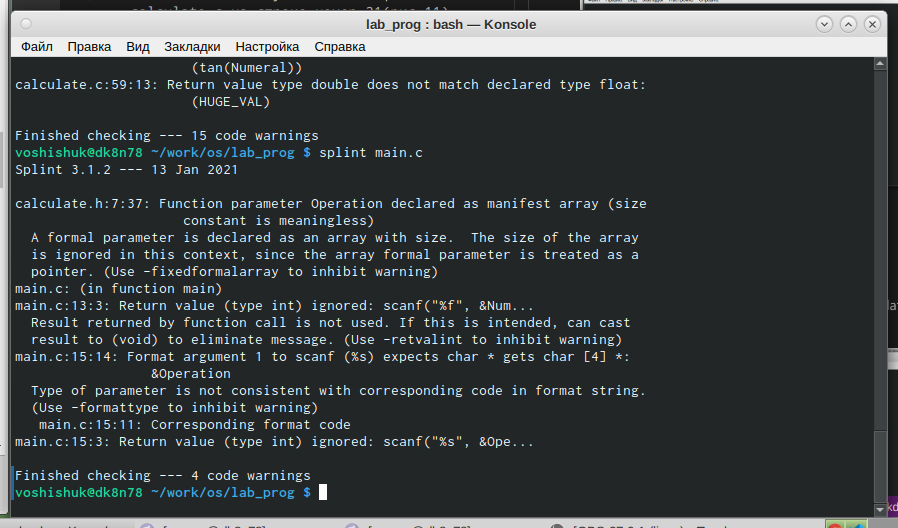


рис.15

# Выводы

Я приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания калькулятора с простейшими функциями.

# Контрольные вопросы

1. Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.
2. Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

* планирование, включающее сбор и анализ требований кфункционалу и другим характеристикам разрабатываемого приложения;
* проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
* непосредственная разработка приложения: o кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; o сборка, компиляция и разработка исполняемого модуля; o тестирование и отладка, сохранение произведённых изменений;
* документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

1. Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C − как файлы на языке C++, а файлы c расширением .o считаются объектными. Например, в команде «gcc -c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль − файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».
2. Основное назначение компилятора языка Си в UNIX заключается вкомпиляции всей программы и получении исполняемого файла/модуля.
3. Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
4. Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefile имеет следующий синтаксис: … : … <команда 1> … Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды − собственно действия, которые необходимо выполнить для достижения цели. Общий синтаксис Makefile имеет вид: target1 [target2…]:[:] [dependment1…] [(tab)commands] [#commentary] [(tab)commands] [#commentary] Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться водной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса Makefile: