# Российский университет дружбы народов

### Факультет физико-математических и естественных наук

## Отчёт по лабораторной работе №12

### Программирование в командном процессоре ОС UNIX. Ветвления и циклы

**Дисциплина:** Операционные системы

**Студент:** Оразгелдиева Огулнур

**Группа:** НПИбд-02-20

**Студ. номер:** 1032205431

2021, Москва

**Лабораторная работа №12**

**Программирование в командном процессоре ОС UNIX. Ветвления и циклы**

**Цель:**

* изучить основы программирования в оболочке ОС UNIX.
* научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов

**Задачи:**

1. Ознакомиться с теоретическим материалом
2. Написать командный файл, используя команды getopts grep
3. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю
4. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N. Число файлов, которые необходимо создать, передаётся в аргументы командной строки.
5. Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории

**Теоретические сведения** [[1]](https://esystem.rudn.ru/pluginfile.php/1142377/mod_resource/content/2/008-lab_shell_prog_1.pdf)

Весьма необходимой при программировании является команда getopts, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используетсядляобъявленияпеременных.Синтаксискомандыследующий: *getopts option-string variable [arg ... ]*

Функция getopts включает две специальные переменные среды — OPTARG и OPTIND. Если ожидается дополнительное значение, то OPTARG устанавливается в значениеэтогоаргумента(будетравнаfile\_in.txtдляопцииiиfile\_out.doc дляопцииo.OPTINDявляетсячисловыминдексомнаупомянутыйаргумент. Функция getopts также понимает переменные типа массив, следовательно, можноиспользоватьеёвфункциинетолькодлясинтаксическогоанализааргументовфункций,ноидляанализавведённыхпользователемданных.

В обобщённой форме оператор цикла for выглядит следующим образом:

*for имя [in список-значений]*

*do список-команд*

*done*

При каждом следующем выполнении оператора цикла for переменная имя принимает следующее значение из списка значений, задаваемых списком список значений. Вообще говоря, список-значений является необязательным. При его отсутствии операторциклаforвыполняется для всехпозиционных параметров или, иначе говоря, аргументов. Таким образом, оператор for i эквивалентен оператору for i in $\*. Выполнение оператора цикла for завершается, когда список значений будет исчерпан.

В обобщённой форме условный оператор if выглядит следующим образом:

*if список-команд*

*then список-команд*

*{elif список-команд then список-команд}*

*[else список-команд]*

*fi*

Выполнение условного оператора if сводится к тому, что сначала выполняетсяпоследовательностькоманд(операторов),которуюзадаётсписок-командвстроке, содержащей служебное слово if. Затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово then. Фраза elif проверяется в том случае, когда предыдущая проверка была ложной. Строка, содержащая служебное слов else, является необязательной.

**Ход работы**

1. Ознакомилась с теоретическим материалом из лабораторной работы 11.

Создала и открыла командный файл с помощью текстового редактора vi. (см. рис. 1)

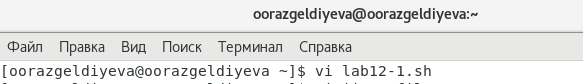


Рисунок 1. Создание командного файла

Написала программу (см. рис. 2), используя getopts и цикл if, который который анализирует командную строку с ключами:

*iinputfile* — прочитать данные из указанного файла;

*ooutputfile* — вывести данные в указанный файл;

*pшаблон* — указать шаблон для поиска;

*-C* — различать большие и малые буквы;

*-n* — выдавать номера строк,

а затем ищет в указанном файле нужные строки, определяемые ключом -p.



Рисунок 2. Программа к заданию 1

**Пояснения:** сначала используем getopts и задаем нужные нам опции (o, i, p, C, n); затем используя цикл if, ищем внутри файла для чтения строки с шаблоном и записываем эту строку в файл для записи, используя команду grep (для этого изучила особенности этой команды в доп. источнике [[2]](https://wiki.merionet.ru/servernye-resheniya/39/rukovodstvo-po-komande-grep-v-linux/).

Создаем файл для чтения (iinputfile.txt) и записываем в него некоторый текст с помощью текстового редактора vi. (см. рис. 3-4)

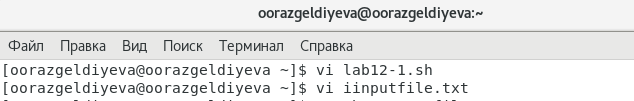


Рисунок 3. Создание файла чтения

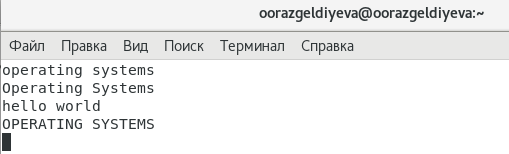


Рисунок 4. Текст для чтения

Создаём файл для записи с помощью команды touch. (см. рис. 5)

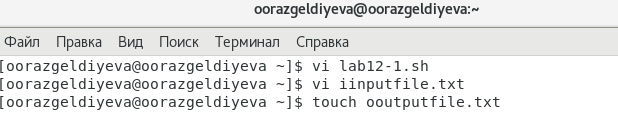


Рисунок 5. Создание файла записи

Предоставляем права на выполнение командного файла с помощью команды *chmod +x*. (см. рис. 6)

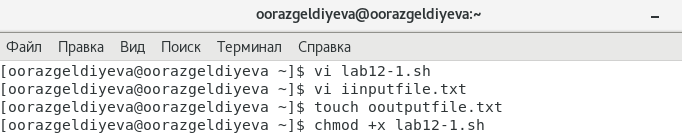


Рисунок 6. Права на выполнение

Выполняем командный файл, в качестве шаблона (слова для поиска) берем слово "systems". После выполнения смотрим содержимое файла записи с помощью cat. (см. рис. 7)

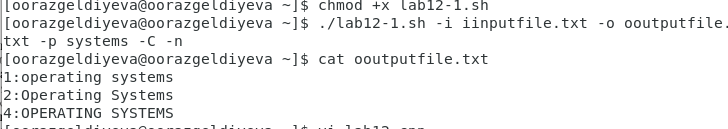


Рисунок 7. Выполнение командного файла

Как видно по рис. 7, в файл записались строки с указанным словом, при этом учитывались заглавные и строчные буквы (опция -С) и номера строк в исходном файле (опция -n).

1. Создаём и открываем файл расширение ".cpp", чтобы записать в него программу на языке С, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции exit(n), передавая информацию в о коде завершения в оболочку. (см. рис. 8)

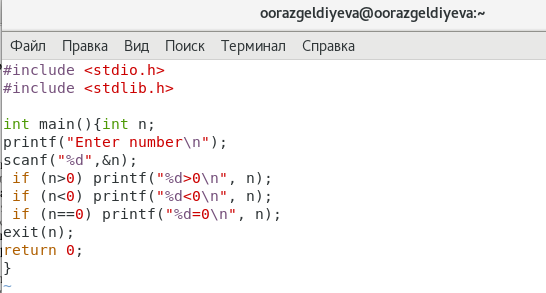


Рисунок 8. Программа на С

**Пояснения:** в первых строчках пишем названия библиотек, которые нужны для выполнения программы (#include...); затем переходим к главной функции: запрос на ввод числа (printf), чтение этого числа при помощи scnaf. Используем циклы if для проверки вводимого числа: больше/меньше/равно 0.

Теперь создаем командный файл с помощью редактора vi, который должен вызывать эту программу и, проанализировав с помощью команды *$?*, выдать сообщение о том, какое число было введено. (см. рис. 9-10)

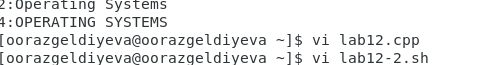


Рисунок 9. Создание командного файла

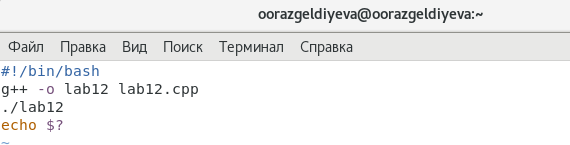


Рисунок 10. Командный файл

Предоставляем командному файлу (lab12-2.sh) и файлу с программой на С (lab12.cpp) права на выполнение. (см. рис. 11)

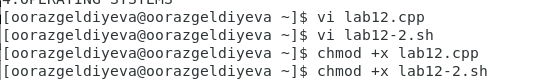


Рисунок 11. Права на выполнение

Выполняем командный файл. Вводим для примера число 9. (см. рис. 12)

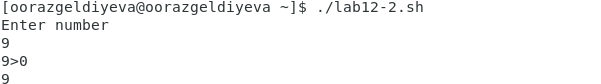


Рисунок 12. Выполнение программы

Теперь для примера введем 0. (см. рис. 13)

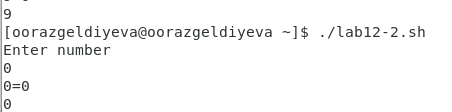


Рисунок 13. Выполнение программы

Программа работает правильно: сначала выводится неравенсто/равентство с 0, потом само введённое число.

1. Создаём командный файл для задания с помощью текстового редактора vi и напишем в нём программу, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки.Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). (см. рис. 14)

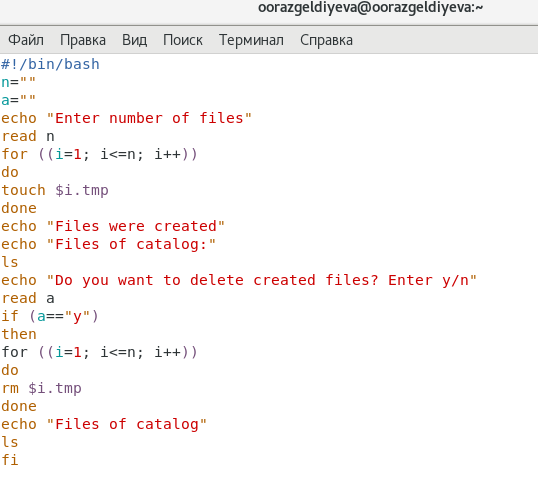


Рисунок 14. Программа к заданию 3

**Пояснения:** создаем переменные n (для вводимого числа - число создаваемых/удаляемых файлов) и a (для ответа). Чтение числа при помощи read; затем при помощи цикла for создаём файлы в текущем каталоге, используя touch, i.tmp (i - номер создаваемого файла). Далее выводим запрос на удаление, если ответ "y", то удаляем созданные файлы. (использовали if).

Выполняем командный файл. Создаем и потом удаляем 5 файлов. (см. рис. 15)

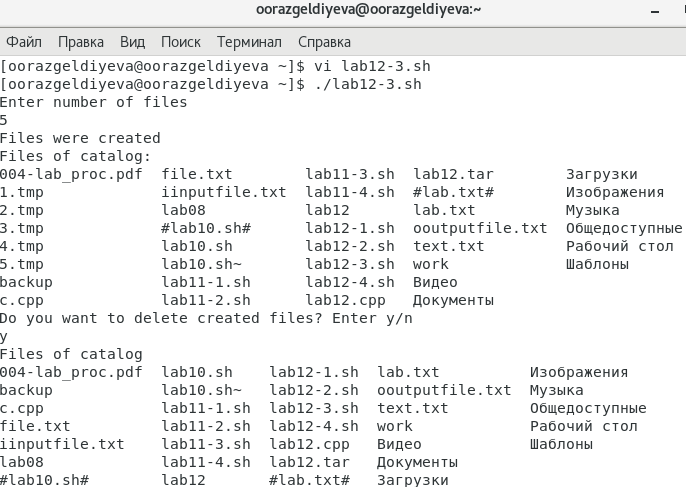


Рисунок 15. Выполнение программы

Как видим, в домашнем каталоге мы создали 5 файлов, затем удалили их.

1. Создаём и открываем командный файл при помощи текстового редактора vi и составляем в нем программу (см. рис. 16), которая с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (используем команду find).

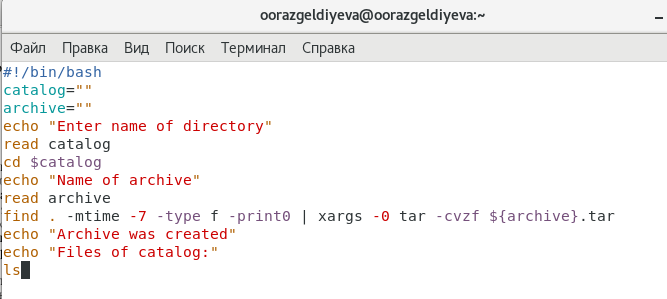


Рисунок 16. Программа к заданию 4

**Пояснения:** создаем переменные *catalog* и *archive*, которые будут содержать название задаваемого каталога и создаваемого архива соответственно. Читаем название каталога и переходим в него, используя соответственно команды *read* и*cd*. С помощью команды *find*, конвейера и архиватора *tar* создаем архив и добавляем туда файла данного каталога, которые были изменены неделю тому назад. (для этого изучила особенности команды find в доп. источнике [[3]](https://losst.ru/komanda-find-v-linux).

Предоставляем права на выполнение командного файла lab12-4.sh и выполняем программу. (см. рис. 17)

В домашнем каталоге и его подкаталогах ищем файлы и архивируем их с помощью архиватора tar в архив *lab12.tar*. (см. рис. 17-18)

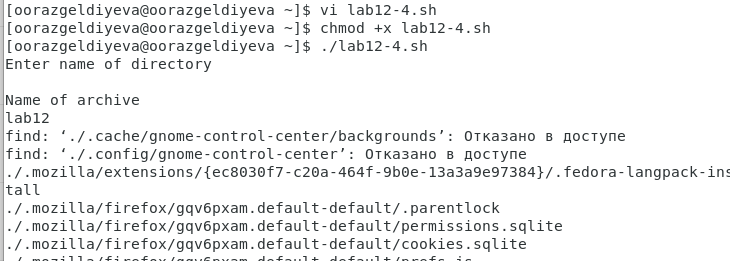


Рисунок 17. Выполнение программы 4

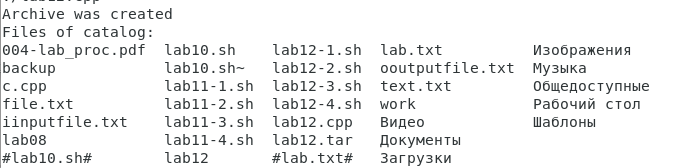


Рисунок 18. Выполнение программы 4

Как видим, в домашнем каталоге появился архив lab12-4.tar.

**Вывод:** на лабораторной работе изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов

**Библиография**

[[1] - РУДН, Операционные системы, Программирование в командном процессоре ОС UNIX. Ветвления и циклы](https://esystem.rudn.ru/pluginfile.php/1142377/mod_resource/content/2/008-lab_shell_prog_1.pdf)

[[2] - Руководство по команде grep ОС Unix/Linux](https://wiki.merionet.ru/servernye-resheniya/39/rukovodstvo-po-komande-grep-v-linux/)

[[3] - Команда find в ОС Unix/Linux](https://losst.ru/komanda-find-v-linux)