Лабораторная работа №11

НПИбд-03-23

Дмитриев Владимир Консатнтинович

Содержание

# 1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# 2 Задание

1. Используя команды getopts grep, написать командный файл, 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. 2. Написать на языке Си программу, 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. Команд- ный файл должен вызывать эту программу и, проанализировав с помощью команды $?, выдать сообщение о том, какое число было введено.

1. Написать командный файл, 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 ко- мандный файл должен уметь удалять все созданные им файлы (если они существуют).
2. Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).

# 3 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

* оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
* С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
* оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
* BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash (в разделе “Выполнение лабораторной работы”). В других оболочках большинство команд будет совпадать с описанными ниже.

# 4 Выполнение лабораторной работы

На первом этапе выполнения работы я создал отдельную директорию для дальнейшей работы, командный файл programm1.sh, текстовый файлы input.txt и output.txt и приступил к написанию файла c командами getopts и grep, 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 (рис. [[1](#fig:001)]) - (рис. [[2](#fig:002)]):

#! /bin/bash  
  
while getopts i:o:p:cn optletter  
do  
case $optletter in  
 i) iflag=1; ival=$OPTARG;;  
 o) oflag=1; oval=$OPTARG;;  
 p) pflag=1; pval=$OPTARG;;  
 c) cflag=1;;  
 n) nflag=1;;  
 \*) echo Illegal option $optletter;;  
 esac  
done  
  
if ! test $cflag  
 then  
 cf=-i  
fi  
  
if test $nflag  
 then  
 nf=-n  
fi  
  
grep $cf $nf $pval $ival >> $oval

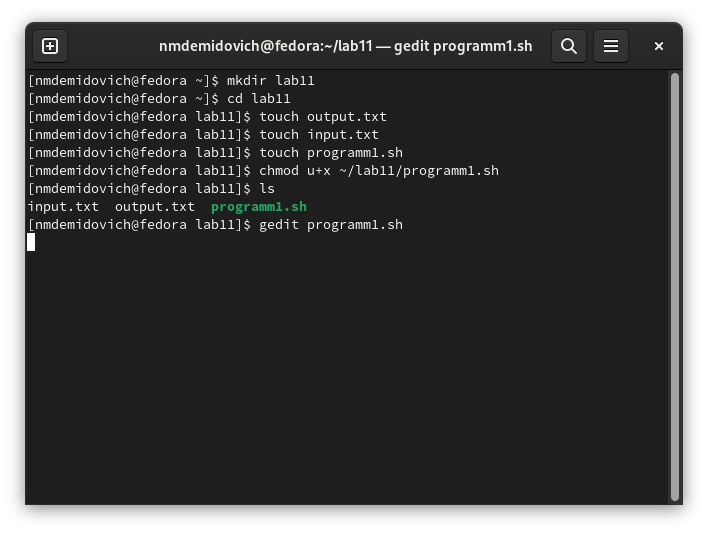


Figure 1: Создание директории lab11, файлов input.txt, output.txt и исполняемого файла programm1.sh

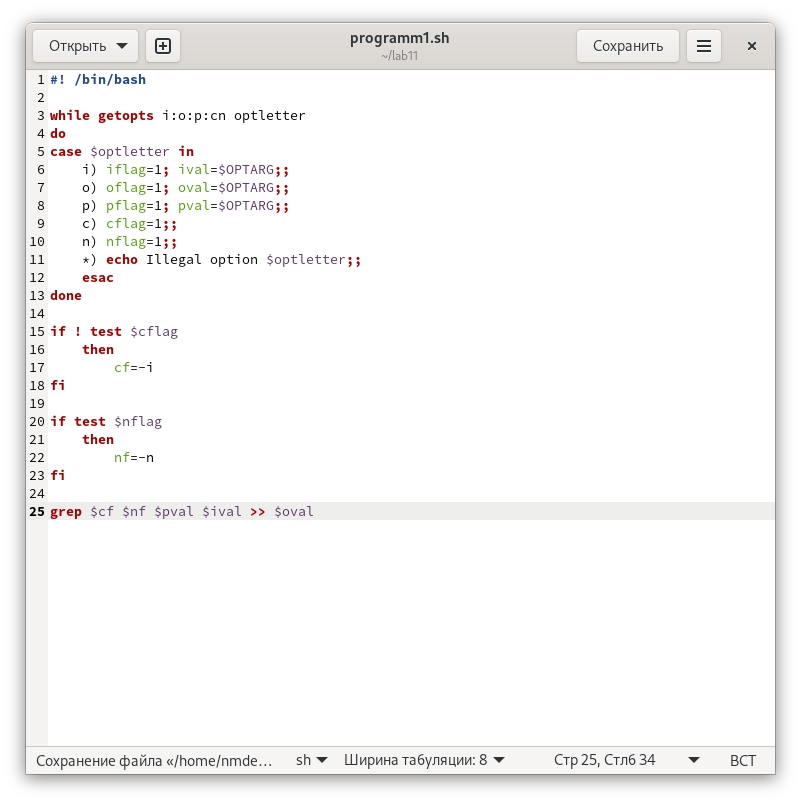


Figure 2: Исполняемый файл programm1.sh

Затем я проверил работу исполняемого файла (рис. [[3](#fig:003)]) - (рис. [[4](#fig:004)]):

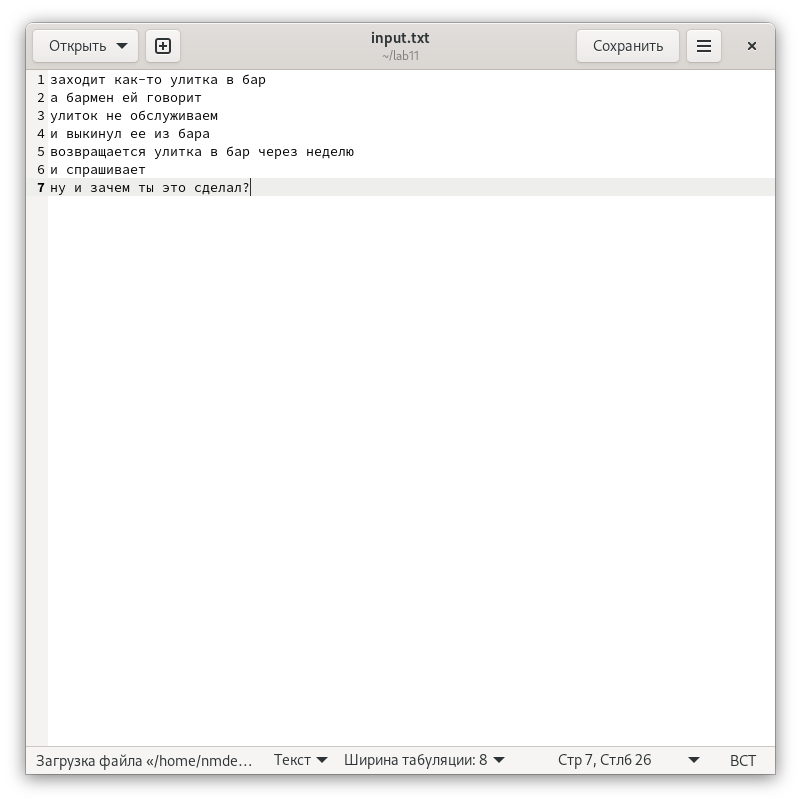


Figure 3: Файл input.txt после работы исполняемого файла programm1.sh

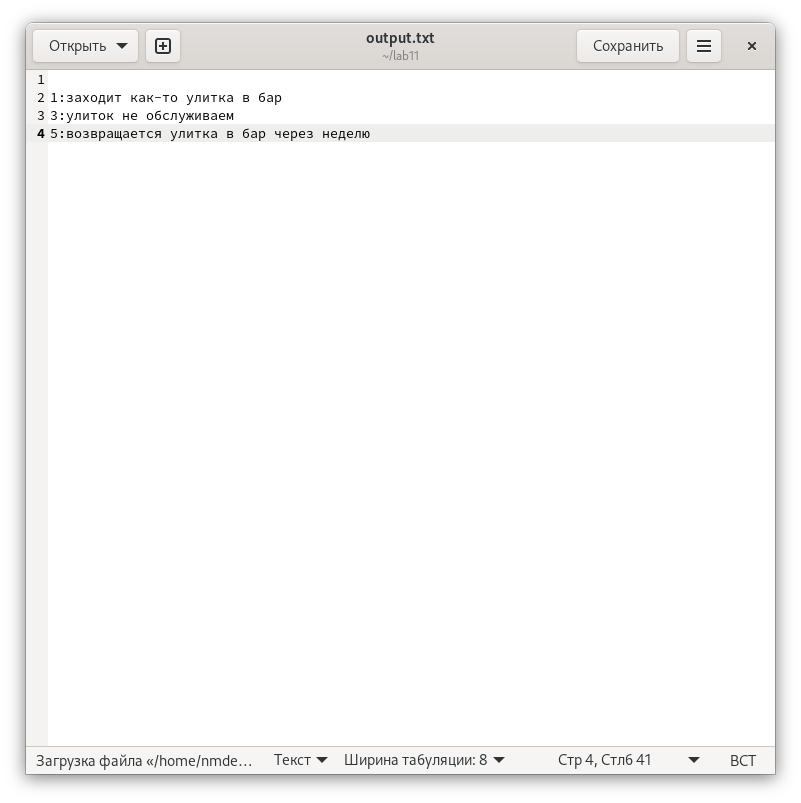


Figure 4: Файл output.txt после работы исполняемого файла programm1.sh

После этого я создал исполняемый файл для второй программы:

#! /bin/bash  
  
gcc -o cprog 12.c  
./cprog  
case $? in  
0) echo "Число равно нулю";;  
1) echo "Число больше нуля";;  
2) echo "Число меньше нуля";;  
esac

А также файл 12.с для программы на языке Си (рис. [[5](#fig:005)]) - (рис. [[9](#fig:009)]):

#include <stdlib.h>  
#include <stdio.h>  
  
int main () {  
 int n;  
 printf ("Введите число: ");  
 scanf ("%d", &n);  
 if(n>0){  
 exit(1);  
 }  
 else if (n==0) {  
 exit(0);  
 }  
 else {  
 exit(2);  
 }

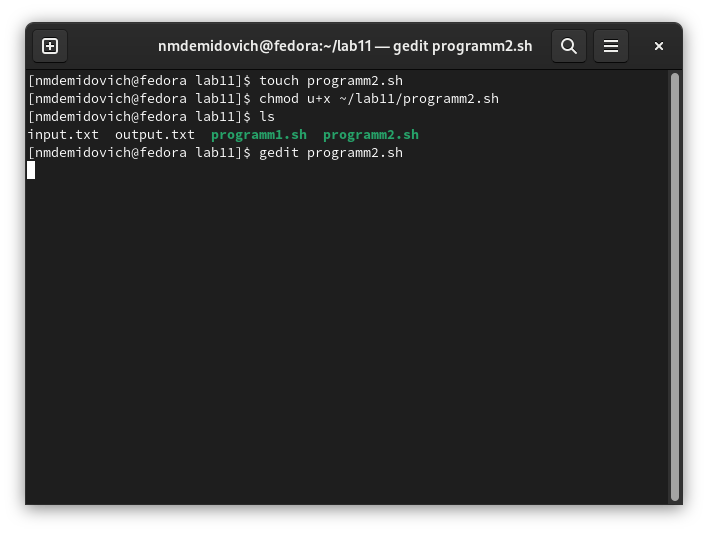


Figure 5: Создание исполняемого файла programm2.sh

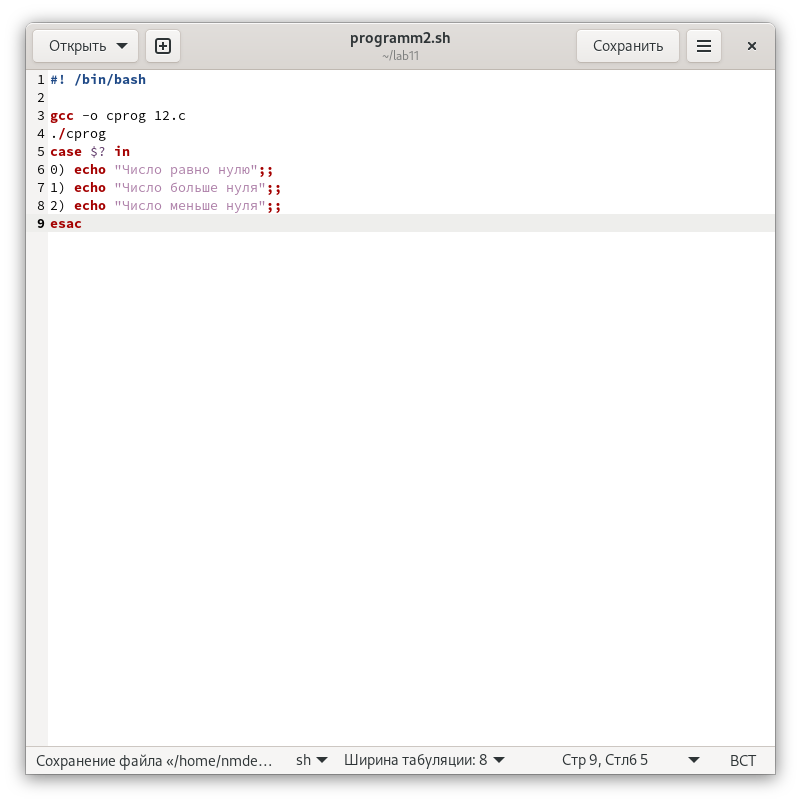


Figure 6: Исполняемый файл programm2.sh

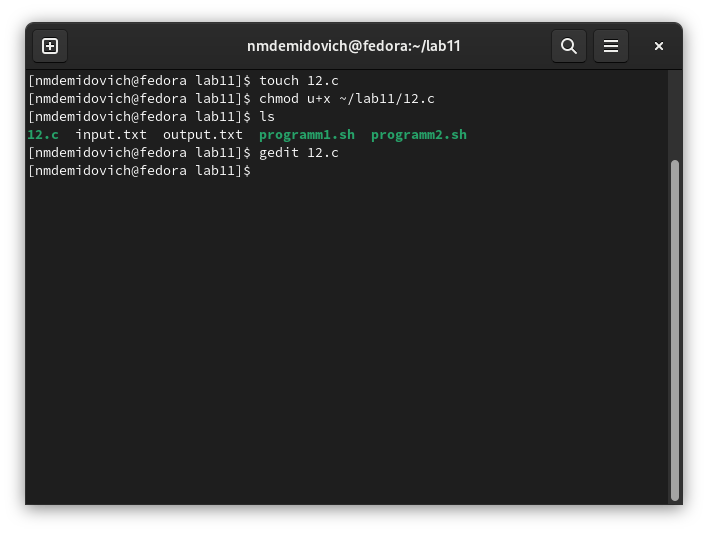


Figure 7: Создание исполняемого файла 12.c

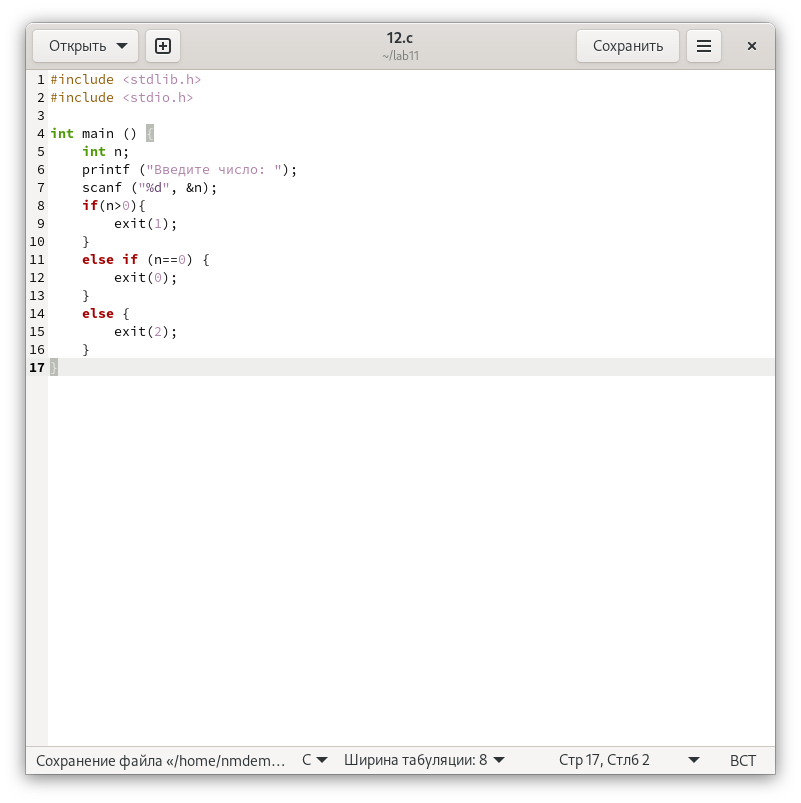


Figure 8: Исполняемый файл 12.c

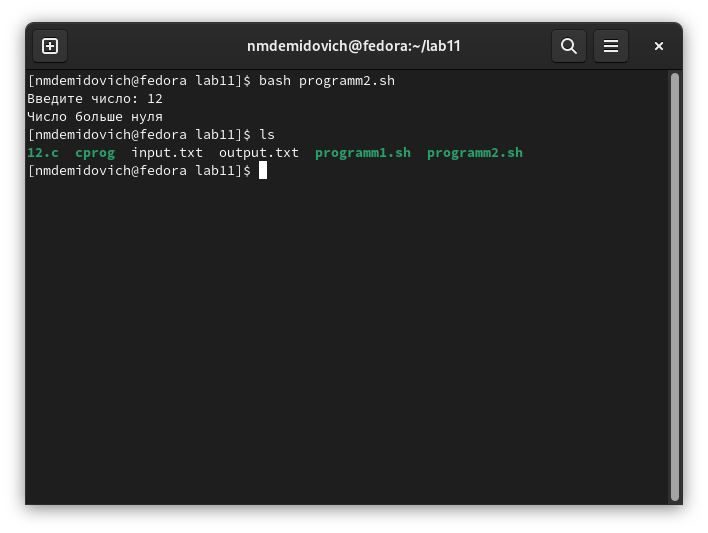


Figure 9: Работа исполняемого файла programm2.sh

Далее я написал командный файл programm3.sh, 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, которое передается в аргументы командной строки (рис. [[10](#fig:010)]) - (рис. [[12](#fig:012)]):

#! /bin/bash  
for((i=1; i<=$\*; i++))  
do  
if test -f "$i".tmp  
then rm "$i".tmp  
else touch "$i.tmp"  
fi  
done

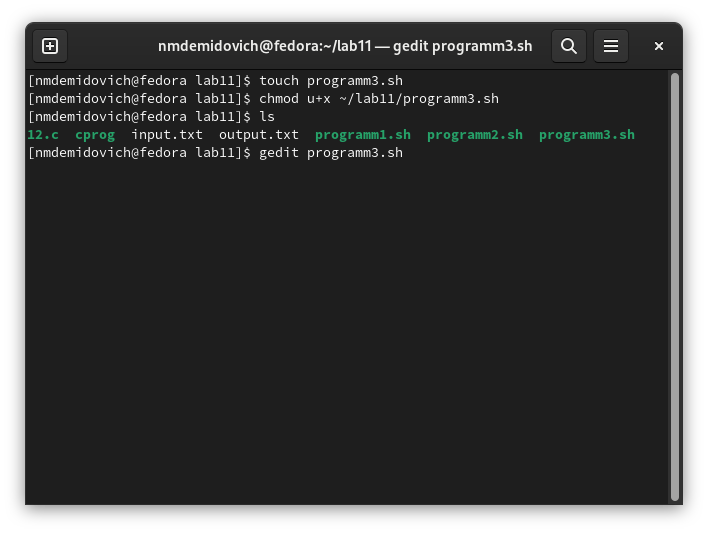


Figure 10: Создание исполняемого файла programm3.sh

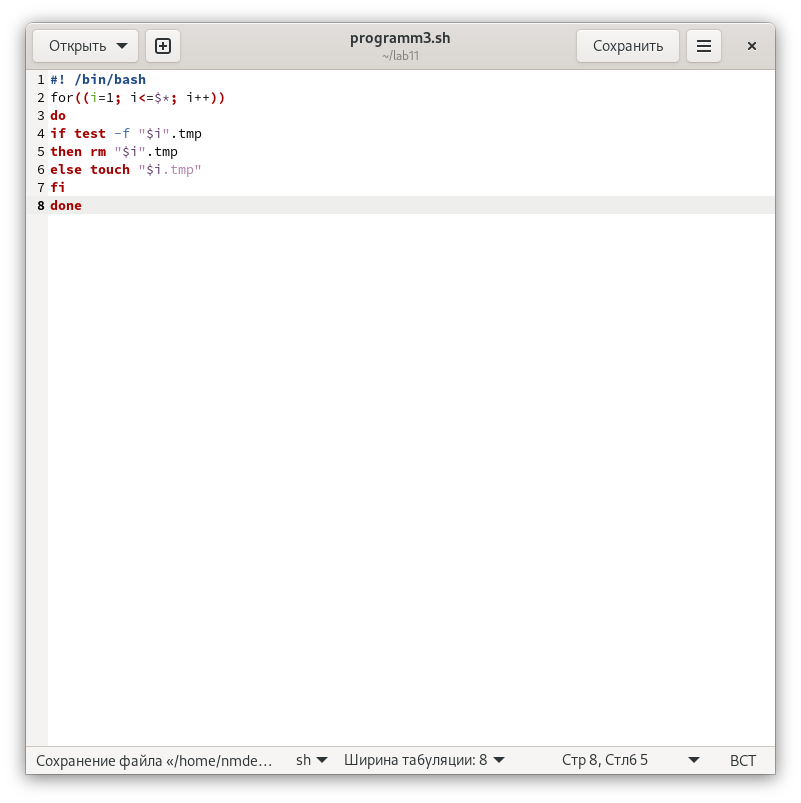


Figure 11: Исполняемый файл programm3.sh

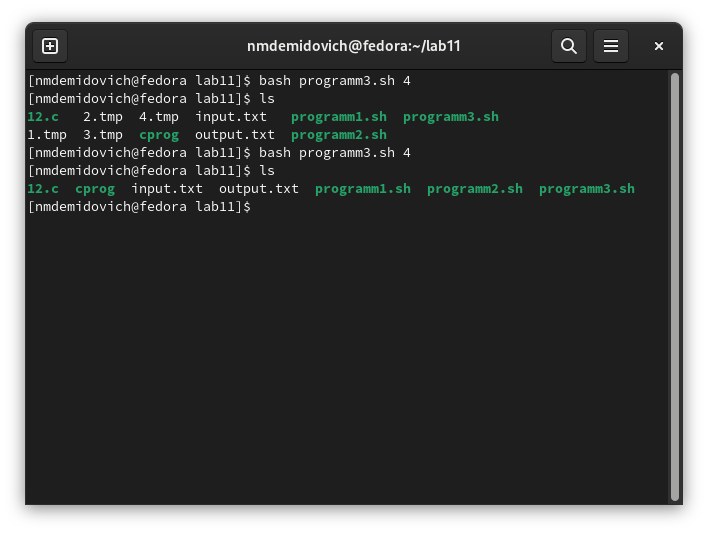


Figure 12: Работа исполняемого файла programm3.sh

И на финальном этапе выполнения работы я создал исполняемый файл для четвертой программы. Это командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории (рис. [[13](#fig:013)]) - (рис. [[15](#fig:015)]):

#! /bin/bash  
find $\* -mtime -7 -mtime +0 -type f > FILES.txt  
tar -cf archive.tar -T FILES.txt

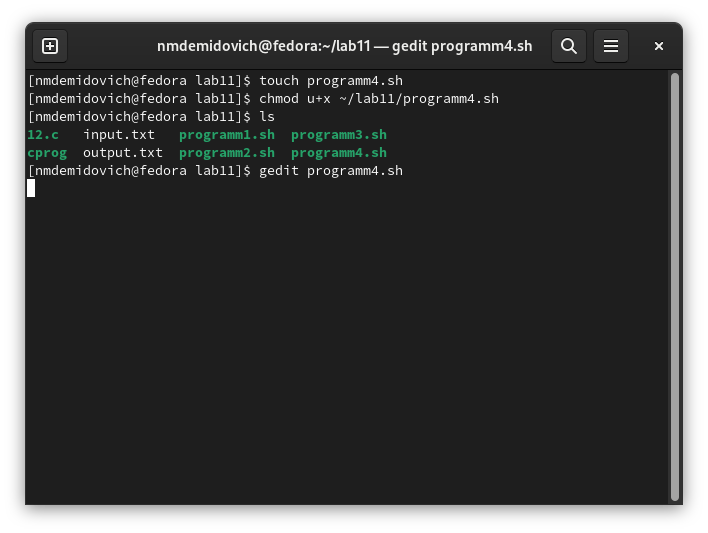


Figure 13: Создание исполняемого файла programm4.sh



Figure 14: Исполняемый файл programm4.sh

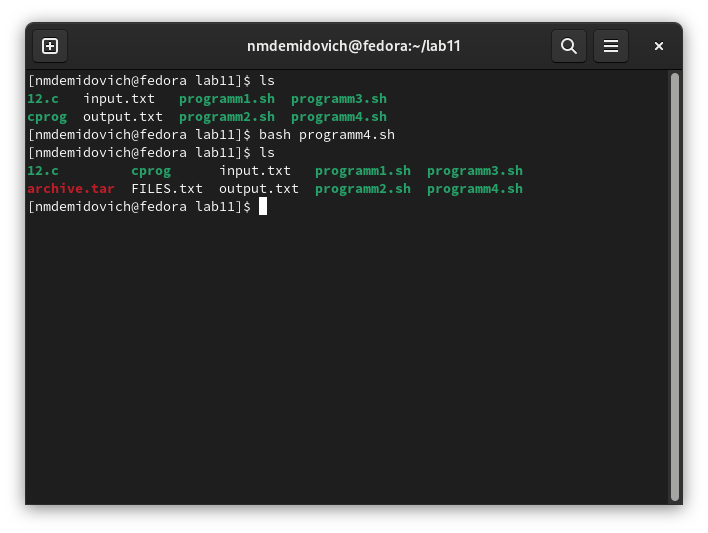


Figure 15: Работа исполняемого файла programm4.sh

# 5 Контрольные вопросы

1. Каково предназначение команды getopts?

Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: getopts option-string variable. Флаги – это опции командной строки, обычно помеченные знаком минус; Например, -F является флагом для команды ls -F. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций option-string — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда getopts может распознать аргумент, она возвращает истину. Принято включать getopts в цикл while и анализировать введенные данные с помощью оператора case. Предположим, необходимо распознать командную строку следующего формата: testprog -ifile\_in.txt -ofile\_out.doc -L -t -r Вот как выглядит использование оператора getopts в этом случае: while getopts o:i:Ltr optletter do case OPTARG;; i) iflag=1; ival=$OPTARG;; L) Lflag=1;; t) tflag=1;; r) rflag=1;; \*) echo Illegal option $optletter esac done Функция getopts включает две специальные переменные среды – OPTARG и OPTIND. Если ожидается дополнительное значение, то OPTARG устанавливается в значение этого аргумента (будет равна file\_in.txt для опции i и file\_out.doc для опции o) . OPTIND является числовым индексом на упомянутый аргумент. Функция getopts также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

1. Какое отношение метасимволы имеют к генерации имён файлов?

При перечислении имён файлов текущего каталога можно использовать следующие символы: − соответствует произвольной, в том числе и пустой строке; ? − соответствует любому одинарному символу; [c1-c2] − соответствует любому символу, лексикографически находящемуся между символами с1 и с2. Например, echo \* − выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls; ls .c − выведет все файлы с последними двумя символами, совпадающими с .c. echo prog.? − выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog.. [a-z] − соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

1. Какие операторы управления действиями вы знаете?

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов проверки некоторого условия. Для решения подобных задач язык программирования bash предоставляет возможность использовать такие управляющие конструкции, как for, case, if и while. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования bash. Поэтому при описании языка программирования bash термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда test, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.

1. Какие операторы используются для прерывания цикла?

Два несложных способа позволяют вам прерывать циклы в оболочке bash. Команда break завершает выполнение цикла, а команда continue завершает данную итерацию блока операторов. Команда break полезна для завершения цикла while в ситуациях, когда условие перестаёт быть правильным. Команда continue используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

1. Для чего нужны команды false и true?

Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования bash: это команда true, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда false, которая всегда возвращает код завершения, не равный нулю (т. е. ложь).

1. Что означает строка if test -f mani.$s, встреченная в командном файле?

Строка if test -f mani.s/s и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).

1. Объясните различия между конструкциями while и until.

Выполнение оператора цикла while сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово while, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово do, после чего осуществляется безусловный переход на начало оператора цикла while. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово while, возвратит ненулевой код завершения (ложь). При замене в операторе цикла while служебного слова while на until условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла while и оператор цикла until идентичны.

# 6 Выводы

В результате выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX и научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Список литературы

Лабораторная работа №11 (Архитектура ОС)