

Отчёт по лабораторной работе №11

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Митичкина Екатерина Павловна

Содержание

| | |
|--------------------------------------|---|
| Цель работы | 1 |
| Задача | 1 |
| Теоретическое введение: | 2 |
| Выполнение лабораторной работы | 2 |
| Задание 1 | 2 |
| Задание 2 | 4 |
| Задание 3 | 6 |
| Задание 4 | 7 |
| Выводы | 8 |
| Ответы на контрольные вопросы | 8 |

Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Задача

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;
 - `-ршаблон` — указать шаблон для поиска;
 - `-C` — различать большие и малые буквы;
 - `-n` — выдавать номера строк.а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.

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. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до ∞ (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).
4. 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`).

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; - C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; - оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; - BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C 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.

Выполнение лабораторной работы

Задание 1

1. Я создала два файла: prog1.sh и 1.txt (Рис. [-@fig:001]-[-@fig:002])

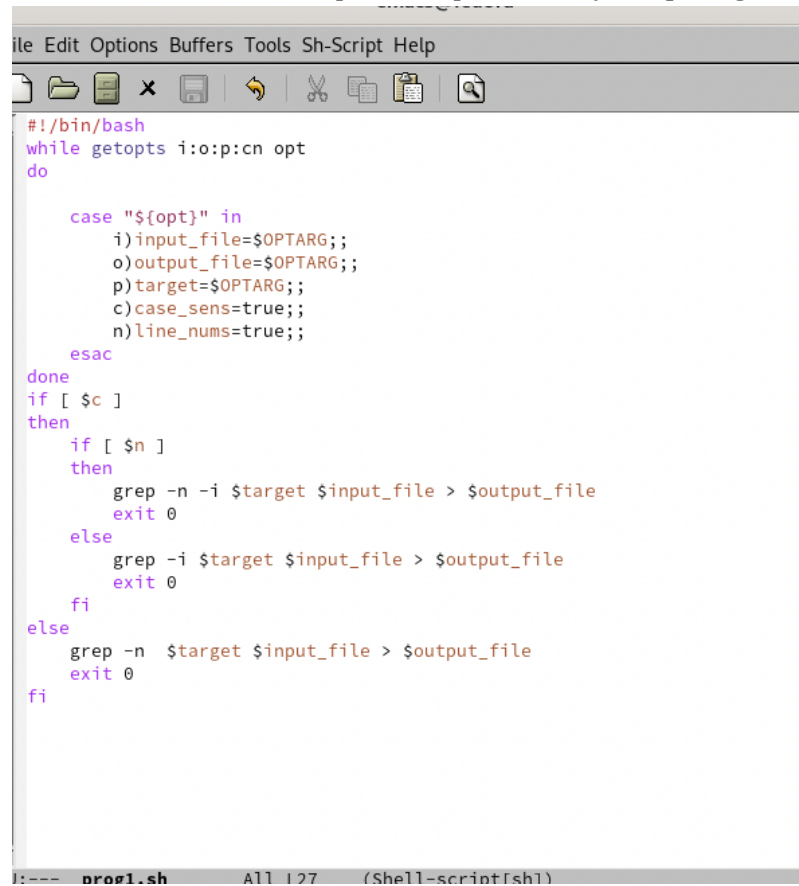
```
epmitichkina@fedora ~  
[epmitichkina@fedora ~]$ touch prog1.sh  
[epmitichkina@fedora ~]$ emacs &  
[1] 4380
```

создание prog1.sh

```
epmitichkina@fedora ~  
[epmitichkina@fedora ~]$ touch 1.txt  
[epmitichkina@fedora ~]$ chmod 755 1.txt
```

создание 1.txt

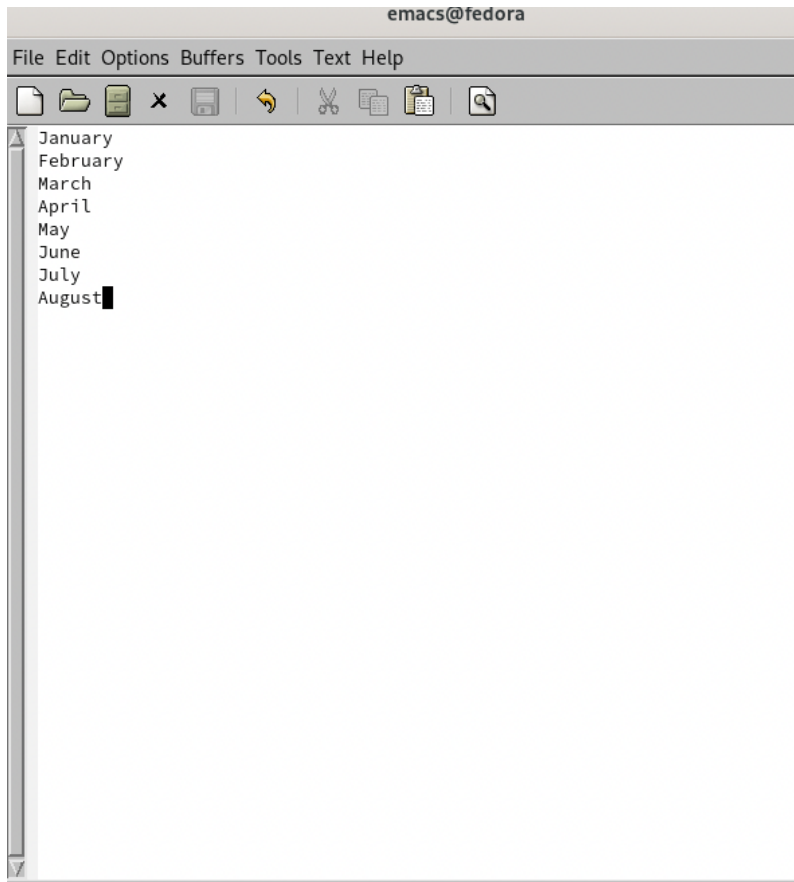
2. Написала код в редакторе emacs (Рис. [-@fig:003]-[-@fig:004])



```
file Edit Options Buffers Tools Sh-Script Help  
#!/bin/bash  
while getopts i:o:p:cn opt  
do  
    case "${opt}" in  
        i)input_file=$OPTARG;;  
        o)output_file=$OPTARG;;  
        p)target=$OPTARG;;  
        c)case_sens=true;;  
        n)line_nums=true;;  
    esac  
done  
if [ $c ]  
then  
    if [ $n ]  
    then  
        grep -n -i $target $input_file > $output_file  
        exit 0  
    else  
        grep -i $target $input_file > $output_file  
        exit 0  
    fi  
else  
    grep -n $target $input_file > $output_file  
    exit 0  
fi
```

l:--- prog1.sh All 127 (Shell-script[sh])

prog1.sh



1.txt

3. Предоставила право на выполнение и проверка файла (Рис. [-@fig:005])

```
[epmitichkina@fedora ~]$ chmod +x 1.txt
[epmitichkina@fedora ~]$ ./prog1.sh -p "May" -i 1.txt -o 2.txt -c -n
bash: ./prog1.sh: Отказано в доступе
[epmitichkina@fedora ~]$ chmod +x prog1.sh
[epmitichkina@fedora ~]$ ./prog1.sh -p "May" -i 1.txt -o 2.txt -c -n
[epmitichkina@fedora ~]$ cat 2.txt
5:May
[epmitichkina@fedora ~]$ ./prog1.sh -p "Ju" -i 1.txt -o 2.txt -n
[epmitichkina@fedora ~]$ cat 2.txt
6:June
7:July
[epmitichkina@fedora ~]$ ./prog1.sh -p "Ju" -i 1.txt -o 2.txt
[epmitichkina@fedora ~]$ cat 2.txt
6:June
7:July
```

Предоставление прав и проверка

Задание 2

1. Я создала два файла: prog2.sh и prog2.c (Рис. [-@fig:006])

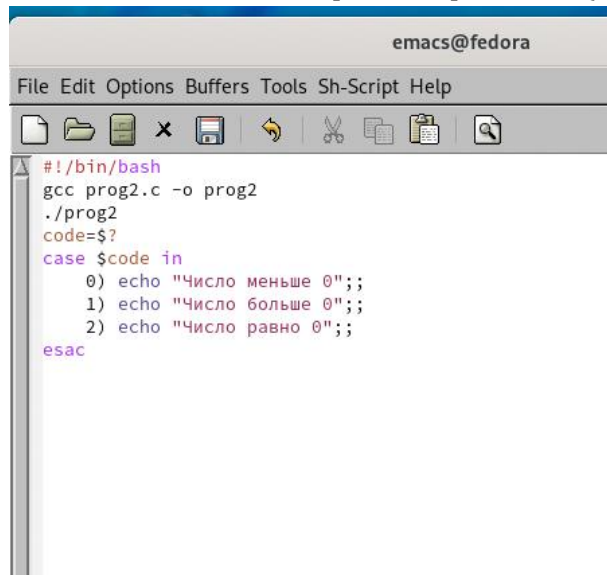
```

[epmitichkina@fedora ~]$ cat prog2.c
[epmitichkina@fedora ~]$ touch prog2.c
[epmitichkina@fedora ~]$ emacs &
[2] 4962
[epmitichkina@fedora ~]$ touch prog2.sh
[1]-  Завершён          emacs
[epmitichkina@fedora ~]$ emacs &
[3] 5058
[epmitichkina@fedora ~]$ chmod +x prog2.sh

```

создание prog2.sh prog2.c

2. Написала код в редакторе emacs (Рис. [-@fig:007]-[-@fig:008])

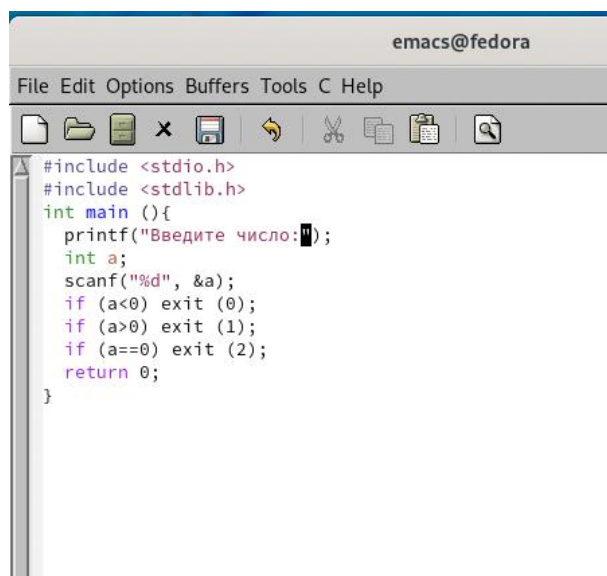


```

#!/bin/bash
gcc prog2.c -o prog2
./prog2
code=$?
case $code in
  0) echo "Число меньше 0";;
  1) echo "Число больше 0";;
  2) echo "Число равно 0";;
esac

```

prog2.sh



```

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

```

prog2.c

3. Предоставила право на выполнение и проверка файла (Рис. [-@fig:009])

```
[1] 5555
[ermitichkina@fedora ~]$ chmod +x prog2.sh
[2]- Завершён      emacs
[ermitichkina@fedora ~]$ ./prog2.sh
Введите число: -89
Число меньше 0
[ermitichkina@fedora ~]$ ./prog2.sh
Введите число: 9
Число больше 0
[ermitichkina@fedora ~]$ ./prog2.sh
Введите число:0
Число равно 0
[ermitichkina@fedora ~]$
```

Предоставление прав и проверка

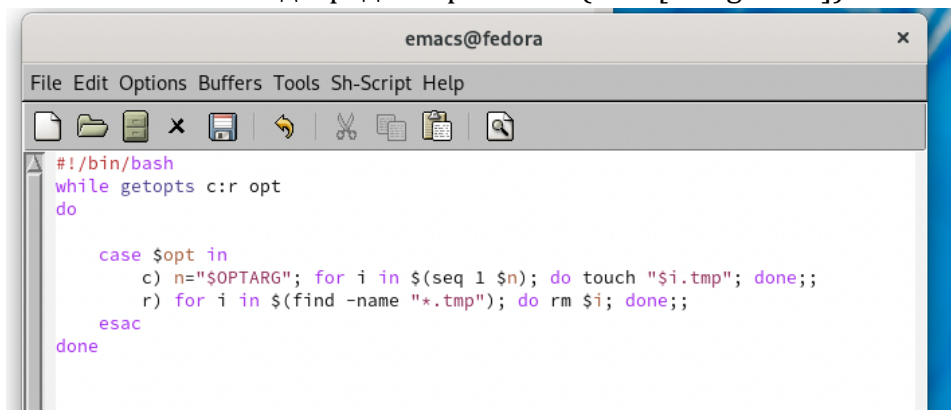
Задание 3

1. Я создала два файла: prog3.sh (Рис. [-@fig:0010])

```
[ermitichkina@fedora ~]$ touch prog3.sh
[ermitichkina@fedora ~]$ emacs &
[4] 5486
```

создание prog3.sh

2. Написала код в редакторе emacs (Рис. [-@fig:0011])



prog3.sh

3. Предоставила право на выполнение и проверка файла (Рис. [-@fig:0012])

```
[epmitichkina@fedora ~]$ chmod +x prog3.sh
[3]- Завершён emacs
[epmitichkina@fedora ~]$ ./prog3.sh -c 5
seq: неверный аргумент с плавающей точкой: «OPTARG»
По команде «seq --help» можно получить дополнительную информацию.
[epmitichkina@fedora ~]$ ./prog3.sh -c 5
[epmitichkina@fedora ~]$ ls
01      2.txt  prog1.sh  prog2.sh  Видео      Общедоступные
1.tmp   3.tmp   prog1.sh~ prog2.sh~  Документы  'Рабочий стол'
1.txt   4.tmp   prog2     prog3.sh  Загрузки   Шаблоны
1.txt~  5       prog2.c   prog3.sh~ Изображения
2.tmp   5.tmp   prog2.c~  work      Музыка
[epmitichkina@fedora ~]$ ./prog3.sh -r
[epmitichkina@fedora ~]$ ls
01      5       prog2.c   prog3.sh  Документы  Общедоступные
1.txt   prog1.sh prog2.c~  prog3.sh~ Загрузки   'Рабочий стол'
1.txt~  prog1.sh~ prog2.sh  work      Изображения Шаблоны
2.txt   prog2     prog2.sh~ Видео      Музыка
[epmitichkina@fedora ~]$
```

Предоставление прав и проверка

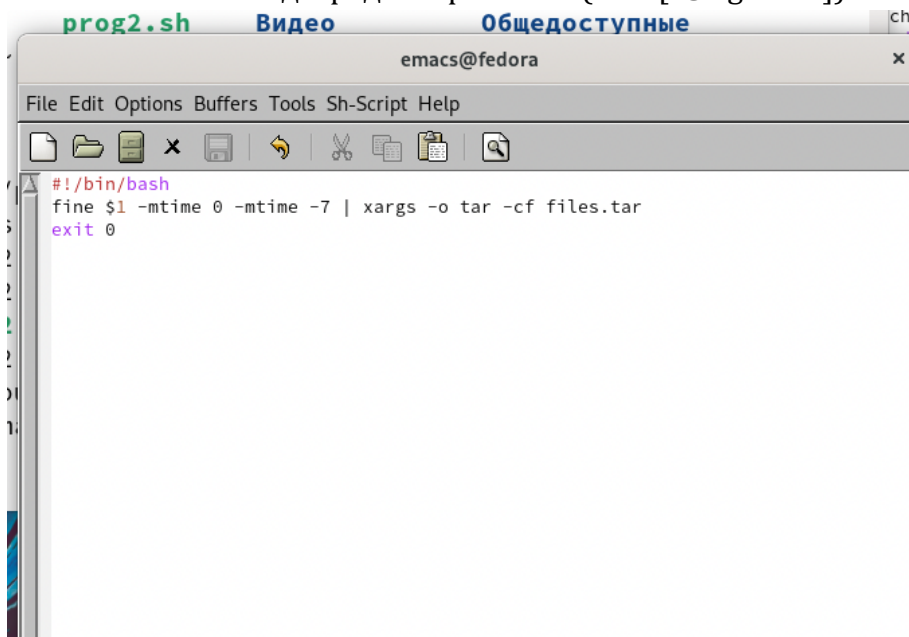
Задание 4

1. Я создала два файла: prog4.sh (Рис. [-@fig:0013])

```
[epmitichkina@fedora ~]$ touch prog4.sh
[epmitichkina@fedora ~]$ emacs &
[5] 5763
```

создание prog4.sh

2. Написала код в редакторе emacs (Рис. [-@fig:0014])



prog4.sh

3. Предоставила право на выполнение и проверка файла (Рис. [-@fig:0015])

```

^C[epmitichkina@fedora ~]$ ./prog4.sh 4
[epmitichkina@fedora ~]$ ls
01      5      prog2      prog3.sh  Видео      Общедоступные
1.txt   files     prog2.c    prog3.sh~ Документы   'Рабочий стол'
1.txt~  files.tar prog2.c~   prog4.sh  Загрузки    Шаблоны
2.txt   prog1.sh  prog2.sh   prog4.sh~ Изображения
4       prog1.sh~ prog2.sh~   work      Музыка
[epmitichkina@fedora ~]$ █

```

Предоставление прав и проверка

Выводы

В результате работы изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Ответы на контрольные вопросы

1. Каково предназначение команды getoptс?

Команда getoptс осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: getoptс option-string variable [arg ...]. Флаги – это опции командной строки, обычно помеченные знаком минус; Например, для команды ls флагом может являться -F. Строка опций option-string – это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда getoptс может распознать аргумент, то она возвращает истину. Принято включать getoptс в цикл while и анализировать введенные данные с помощью оператора case. Функция getoptс включает две специальные переменные среды – OPTARG и OPTIND. Если ожидается дополнительное значение, то OPTARG устанавливается в значение этого аргумента. Функция getoptс также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

2. 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; - [с1-с2] – 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 .с – выведет все файлы с последними двумя символами, совпадающими с .с. - 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.

3. 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.

4. 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.

5. 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). Примеры бесконечных циклов: `while true do echo hello andy done until false do echo hello mike done`

6. Что означает строка `if test -f mans/i.$s`,встреченная в командном файле?

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

7. 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.