Лабораторная работа №11

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Парфенова Елизавета Евгеньевна

Содержание

Список иллюстраций

Список таблиц

1 Цель 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. 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).

3 Выполнение лабораторной работы

В первом задании было необходимо написать командный файл, который анализирует командную строку с ключами и ищет в указанном файле строки с ключом p. Это я осуществила с помощью циклов и команды getopts. Поиск выполнила командой grep. Пишу скрипты я в редакторе emacs, перед выполнением я создаю файл “script1”. (рис. 3.1)

```
#!/bin/bash

while getopts "i:o:p:cn" opt
do
        case $opt in
                i)inputfile="$OPTARG";;
                o)outputfile="$OPTARG";;
                p)sample="$OPTARG";;
                c)reg="";;
                n)line="";;
        esac
done

grep -n "$sample" "$inputfile" > "$outputfile"
```

Рис. 3.1: Код превого командого файла

Литсинг первого скрипта:

#!/bin/bash

while getopts “i:o:p:cn” opt

do

```
case $opt in
```

```
        i)inputfile="$OPTARG";;
        o)outputfile="$OPTARG";;
        p)sample="$OPTARG";;
        c)reg="";;
        n)line="";;
esac
```

done

grep -n “$sample""inputfile” > “$outputfile”

После предоставляю право на выполнение командой ***chmod +x script1*** и запускаю, написав в косоль название скрипта и указав все данные: ***./script1 -i conf.txt -o result.txt -p n etconf -c -n***.Файл conf.txt я взяла с одной из прошлых Лабораторных работ. В результате в папке создается файл с заданным названием и в него перезаписываются все файлы с указанными характеристиками. (рис. 3.2)

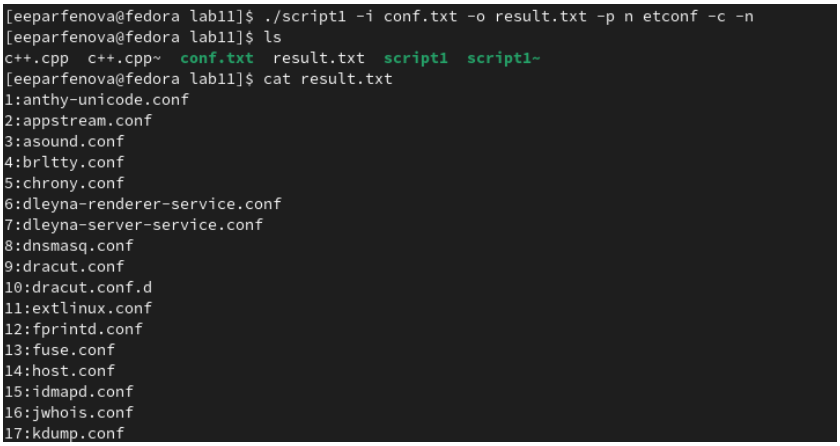
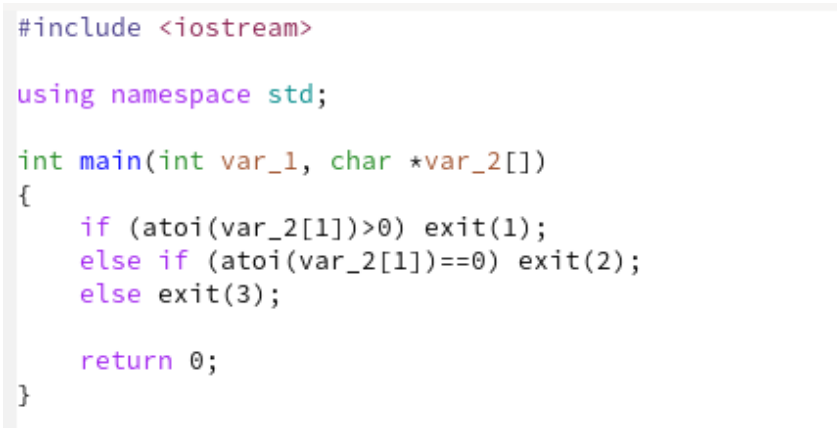


Рис. 3.2: Работа превого командого файла

Приступаем к написанию второго командного файла. Для него я заранее подготовила проограмму на c++, которая опрделяет, больше нуля, меньше нуля или ранво нуля заданное число. (рис. 3.3)



```
#include <iostream>

using namespace std;

int main(int var_1, char *var_2[])
{
    if (atoi(var_2[1])>0) exit(1);
    else if (atoi(var_2[1])==0) exit(2);
    else exit(3);

    return 0;
}
```

Рис. 3.3: Код программы на C++

Листинг программы :

#include

using namespace std;

int main(int var_1, char *var_2[])

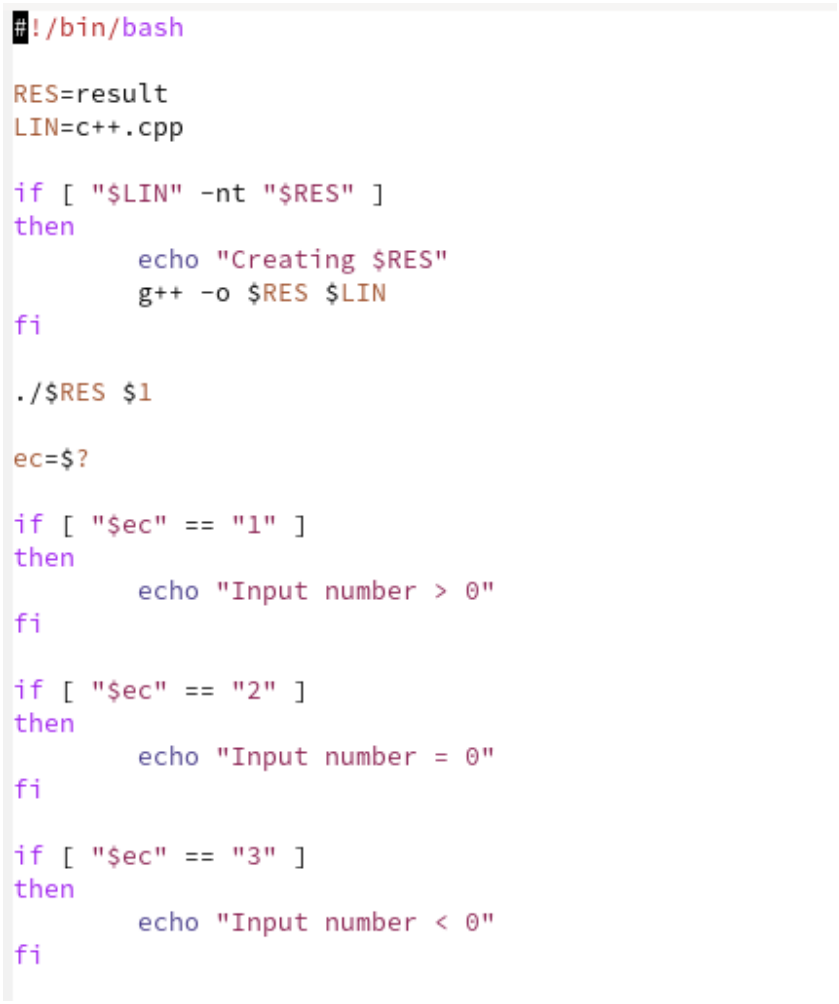
{

```
if (atoi(var_2[1])>0) exit(1);
else if (atoi(var_2[1])==0) exit(2);
else exit(3);


return 0;
```

}

После я создаю файл script2 и осуществляю программу с помощью циклов if. Также использую компилятор g++. (рис. 3.4)



```
#!/bin/bash

RES=result
LIN=c++.cpp

if [ "$LIN" -nt "$RES" ]
then
        echo "Creating $RES"
        g++ -o $RES $LIN
fi

./$RES $1

ec=$?

if [ "$ec" == "1" ]
then
        echo "Input number > 0"
fi

if [ "$ec" == "2" ]
then
        echo "Input number = 0"
fi

if [ "$ec" == "3" ]
then
        echo "Input number < 0"
fi
```

Рис. 3.4: Код второго командного файла

Литсинг второго скрипта:

#!/bin/bash

RES=result

LIN=c++.cpp

if [“$LIN" $-$ nt"RES”]

then

```
echo "Creating $RES"
    g++ -o $RES $LIN
```

fi

./$RES $1
ec=$?
if [“$ec” == “1”]
then

```
echo "Input number > 0"
```

fi
if [“$ec” == “2”]
then

```
echo "Input number = 0"
```

fi
if [“$ec” == “3”]
then

```
echo "Input number < 0"
```

fi

После я делаю файл исполняемым и запускаю его командой ***./script2*** записывая в качестве входных данных любое число. Программа работает успешно и для положительный чисел, и для отрицательных, и для чисел равных нулю. (рис. 3.5)

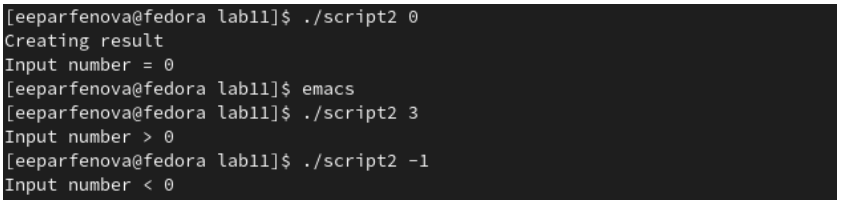


Рис. 3.5: Работа второго командного файла

Начинаем писать третий скрипт. Создала файл “script3”. Я написала код, используя команду getopts и два цикла for. Я выбрала формат tmp, который и был указан в примере. (рис. 3.6)

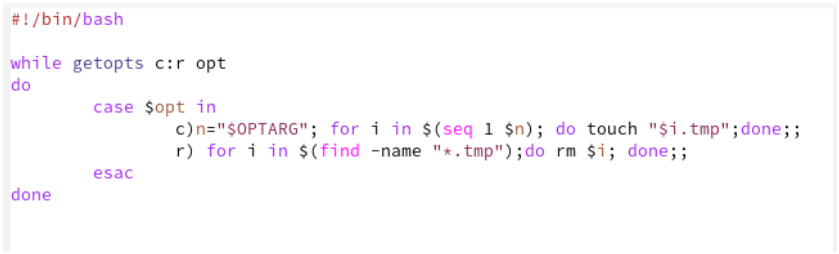


Рис. 3.6: Код третьего командного файла

Литсинг третьего скрипта:

#!/bin/bash

while getopts c:r opt

do

```
case $opt in
        c)n="$OPTARG"; for i in $(seq 1 $n); do touch "$i.tmp";done;;
        r) for i in $(find -name "*.tmp");do rm $i; done;;
esac
```

done

После сделала файл исполняемым и проверила работу скрипта, используя ***./script3*** и одну из опций. Опцией -c я создала 5 файлов нужного формата (число я также ввела после опции) и проверила их создание, а поцией -r я удалила эти файлы и снова проверила успешность команды. Скрипт работает корректно. (рис. 3.7)

Рис. 3.7: Работа третьего командного файла

Приступаем к поеследнему заданию. В нем требовалось заархивировать файлы из указанного каталога, измененные не более, чем неделю назад. Архивировала я с помощью tar. Я снова использовлаа getopts. Вот так выглядит код программы: (рис. 3.8)

Рис. 3.8: Код четвертого командного файла

Литсинг четвертого скрипта:

#!/bin/bash

while getopts :d: opt

do

```
case $opt in
        d)dir="$OPTARG";;
esac
```

done

find $dir -mtime -7 -mtime +0 -type f > archiv.txt

tar -cf result.tar -T archiv.txt

Далее сделала файл исполняемым и запустила скрипт командой ***./script4***. Я взяла домашнюю директорию. Прграмма долго выполнялась из-за большого объема файлов, но в итоге был создан архив с заданным названием, в котором находились все файлы домашнего каталога, которые были изменены в последнюю неделю. (рис. 3.9) (рис. 3.10)

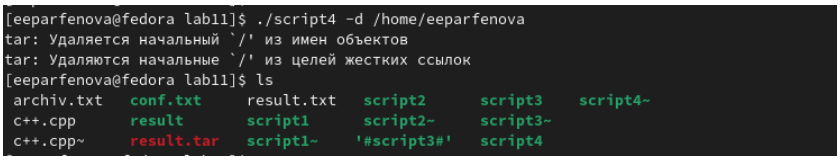


Рис. 3.9: Работа четвертого командного файла

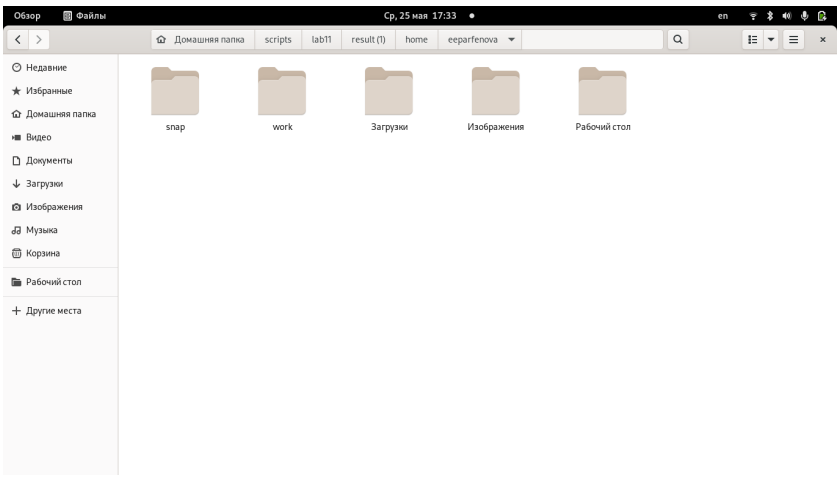


Рис. 3.10: Содержимое созданного архива

4 Выводы

Мы изучили основы программирования в оболочке ОС UNIX и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Контрольные вопросы

1. Каково предназначение команды getopts?

Весьма необходимой при программировании является команда getopts, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: getopts option-string variable [arg ...] 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 $optletterino)oflag = 1; oval =$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.

2. Какое отношение метасимволы имеют к генерации имён файлов?

При перечислении имен файлов текущего каталога можно использовать следующие символы:

· * — соответствует произвольной, в том числе и пустой строке;

· ? — соответствует любому одному символу;

· [c1-c1] — соответствует любому символу, лексикографически на ходящемуся между символами c1 и c2. ·

echo * — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls;

· ls .c — выведет все файлы с последними двумя символами, равными .c.

· echo prog.? — выдаст 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.

Какие операторы управления действиями вы знаете?

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования bash предоставляет Вам 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 дела являются операторами языка программирования bash. Поэтому при описании языка про-

граммирования bash термин оператор будет использоваться наравне с термином команда.

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 перестает быть правильным. Пример бесконечного цикла while, с прерыванием в момент, когда файл перестает существовать: while true do if [! -f $file] then break fi sleep 10 done.

5. Для чего нужны команды false и true?

true : всегда возвращает 0 в качестве кода выхода.
false : всегда возвращает 1 в качестве кода выхода.

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

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. Введенная строка означает условие существования файла man$s$/i.$s

7. Объясните различия между конструкциями while и until.

Если речь идет о 2-х параллельных действиях, то это while. когда мы показываем, что сначала делается 1-е действие. потом оно заканчивается при наступлении 2-го действия, применяем until.