Лабораторная работа № 11

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Шулуужук Айраана Вячеславовна НПИбд-02-22

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; – -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 -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.

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 ко- мандный 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 Теоретическое введение

3.1 Командные процессоры (оболочки)

Командный процессор (командная оболочка, интерпретатор команд shell) — это про- грамма, 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 програм- мой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей сов- мещает 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 опера- ционных 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 других оболоч- ках большинство команд будет совпадать с описанными

ниже.

4 Выполнение лабораторной работы

Используя команды getopts grep, напишем командный файл, который анализирует командную строку с ключами (рис. 4.1)

```
while getopts "i:o:p:c:n" opt
do
case $opt in
i)inputfile="$OPTARG";;
o)outputfile="$OPTARG";;
p)shablon="OPTARG";;
c)registr="";;
n)number="";;
esac
done

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

Рис. 4.1: скрипт 1

Скомпилируем данные файл и проверим его работу (рис. 4.2)

```
[avshuluuzhuk@fedora lab11]$ vi lab11_1
[avshuluuzhuk@fedora lab11]$ ./lab11_1 -i new.txt -o output.txt -p h -c -n
[avshuluuzhuk@fedora lab11]$ ls
lab11_1  new.txt  output.txt
[avshuluuzhuk@fedora lab11]$
```

Рис. 4.2: результат работы командного фпйла

Напишем на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю (рис. 4.3)

```
#include <iostream>
using namespace std;

int main(int argument, char *arg[]){
        if (atoi(arg[1]) > 0){
                exit(1);
        }
        else if (atoi(arg[1]) == 0) {
                exit(2);
        }
        else {
                exit(3);
        }
        return 0;
}
```

Рис. 4.3: программа сравнения чисел на C

Нашишем командный файл который должен вызывать эту программу и, проанализировав с помощью команды $?, выдать сообщение о том, какое число было введено.(рис. 4.4)

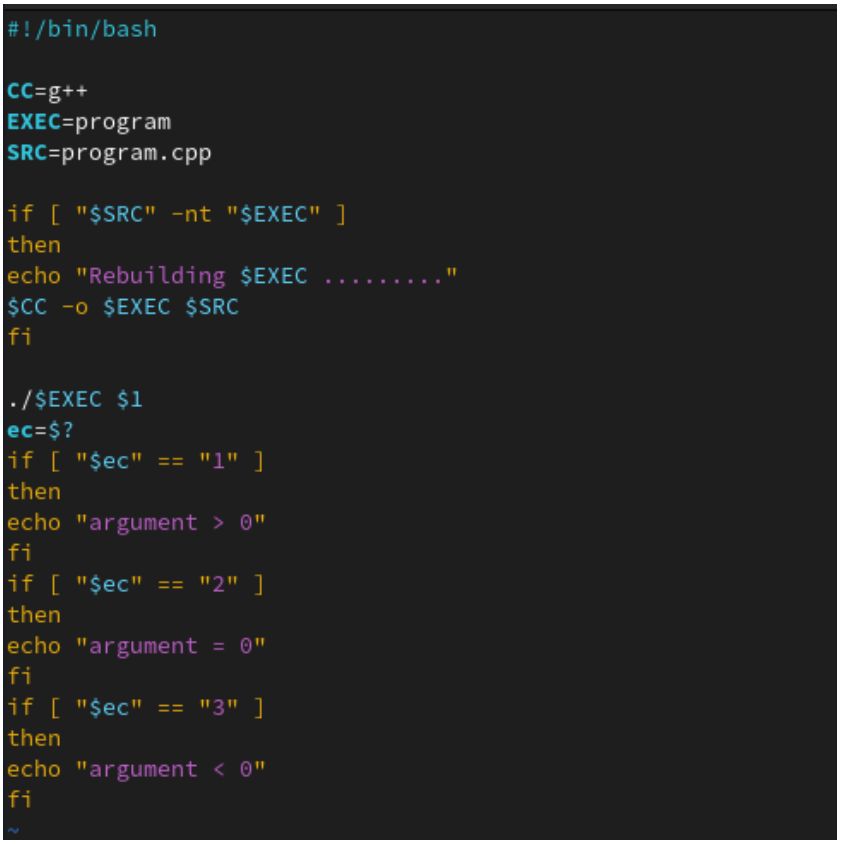

Рис. 4.4: скрипт 2

Проверим работу командного файла, используя число для сравнения в качестве аргумента (рис. 4.5)

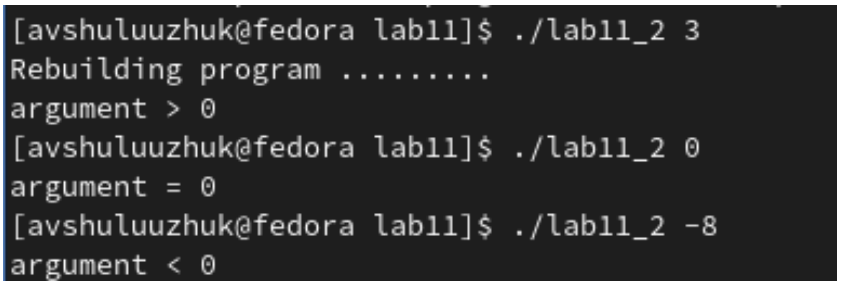

Рис. 4.5: результат запуска скрипта 2

Создаем новый файл lab11_3 для скрипта 3. Напишем командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до n и

удаляющий все созданные им файлы (рис. 4.6)

```
#!/bin/bash
while getopts c:r opt
do
case $opt in
        c)n="$OPRTARG"; for i in $(seq 1 $n); do touch "$i.tmp";done;;
        r)for i in $(find -name "*.tmp"); do rm $i; done;;
esac
done
```

Рис. 4.6: скрипт lab11_3

Запустим этот файл и создадим файл tmp, а также сразу удалим этот файл (рис. 4.7)

```
[avshuluuzhuk@fedora lab11]$ ./lab11_3 -c 1
[avshuluuzhuk@fedora lab11]$ ls
1.tmp  lab11_1  lab11_2  lab11_3  new.txt  output.txt  program  program.cp
[avshuluuzhuk@fedora lab11]$ vi lab11_3
[avshuluuzhuk@fedora lab11]$ ./lab11_3 -r
[avshuluuzhuk@fedora lab11]$ ls
lab11_1  lab11_2  lab11_3  new.txt  output.txt  program  program.cpp  prog
[avshuluuzhuk@fedora lab11]$
```

Рис. 4.7: результат запуска скрипта 3

В файле lab11_4 напишем скрипт, который с помощью команды tar запаковывает в архив все файлы в указанной директории. (рис. 4.8)

```
[#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files" ; do
        file=$(echo "$file" | cut -c 3-)
        listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

Рис. 4.8: скрипт lab11_4

Запустим файл и запакуем архив с файлами каталога lab11 (рис. 4.9)

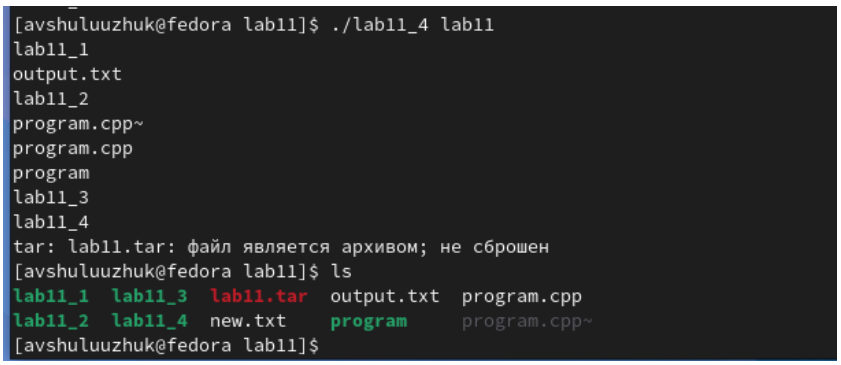

Рис. 4.9: результат запуска скрипта 4

5 Выводы

В ходе выполнения работы мы изучили основы программирования в оболочке ОС UNIX/Linux и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.