

Лабораторная работа №13

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Юсупова Ксения Равиловна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Ответы на контрольные вопросы	13
5	Выводы	16

Список иллюстраций

3.1 код для первой программы	8
3.2 проверили первый код	9
3.3 код для второй программы	9
3.4 второй код для второй программы	10
3.5 Проверили код на работу	10
3.6 код для третьей программы	11
3.7 Проверили код на работу	11
3.8 код для четвертой программы	12
3.9 Проверили код на работу	12

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.

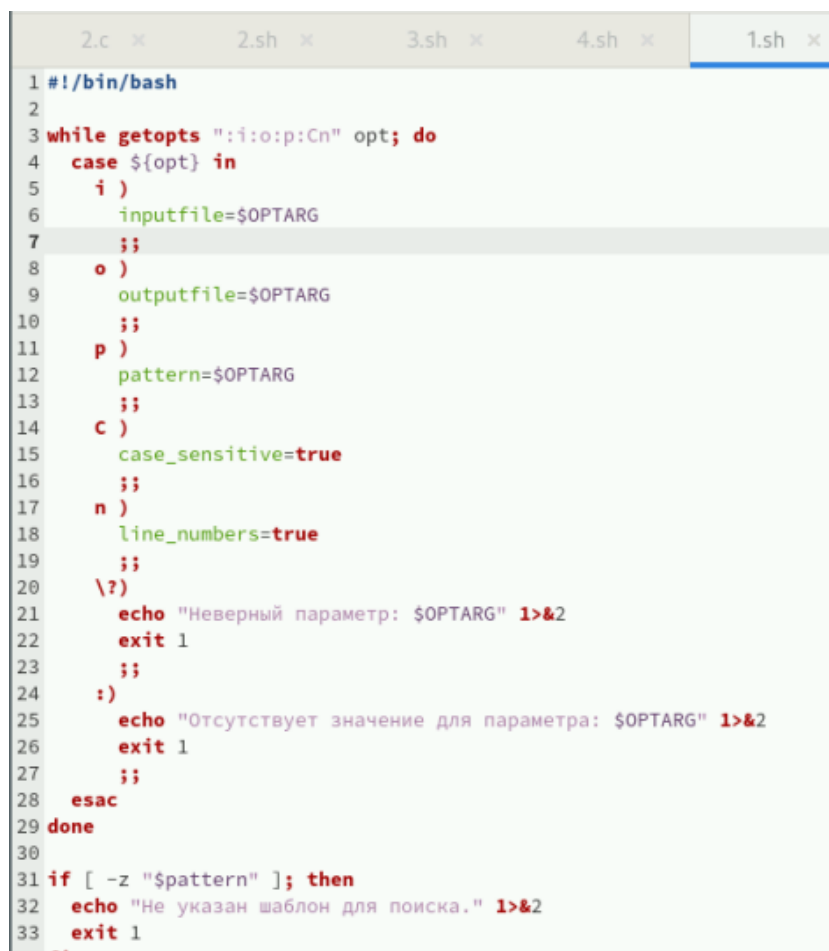
2 Задание

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;
 - `-р`шаблон — указать шаблон для поиска;
 - `-C` — различать большие и малые буквы;
 - `-п` — 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 `tag` 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

Напишем код для первой программы (Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с необходимыми ключами)(рис. 3.1).



```
1 #!/bin/bash
2
3 while getopts ":i:o:p:Cn" opt; do
4     case ${opt} in
5         i )
6             inputfile=$OPTARG
7             ;;
8         o )
9             outputfile=$OPTARG
10            ;;
11        p )
12            pattern=$OPTARG
13            ;;
14        C )
15            case_sensitive=true
16            ;;
17        n )
18            line_numbers=true
19            ;;
20        \?)
21            echo "Неверный параметр: $OPTARG" 1>&2
22            exit 1
23            ;;
24        :)
25            echo "Отсутствует значение для параметра: $OPTARG" 1>&2
26            exit 1
27            ;;
28    esac
29 done
30
31 if [ -z "$pattern" ]; then
32     echo "Не указан шаблон для поиска." 1>&2
33     exit 1
34 fi
```

Рис. 3.1: код для первой программы

Проверили код на работу (рис. 3.2).

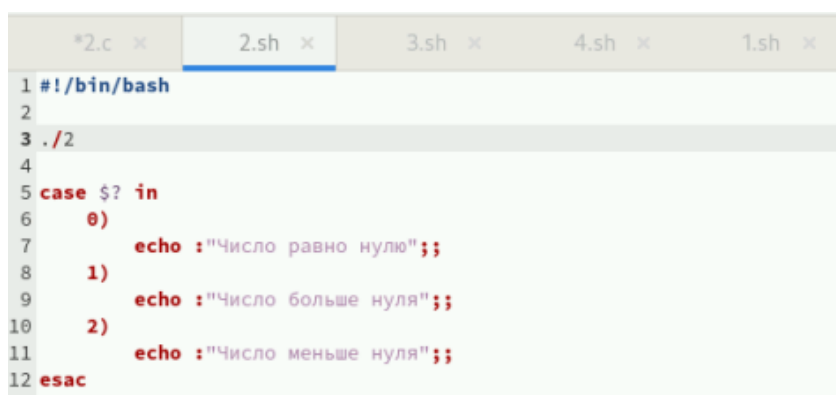

```

[ksyusha@ksyusha lab13]$ ./1.sh -p "У лукоморья" -i text
У лукоморья дуб зелёный;
[ksyusha@ksyusha lab13]$ ./1.sh -p "У лукоморья" -i text -n
1:У лукоморья дуб зелёный;
[ksyusha@ksyusha lab13]$ ./1.sh -p "У Лукоморья" -i text -C
У лукоморья дуб зелёный;

```

Рис. 3.2: проверили первый код

Напишем код для второй программы (Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.)(рис. 3.3).



```

2.sh x
1 #!/bin/bash
2
3 ./2
4
5 case $? in
6   0)
7     echo : "Число равно нулю";;
8   1)
9     echo : "Число больше нуля";;
10  2)
11    echo : "Число меньше нуля";;
12 esac

```

Рис. 3.3: код для второй программы

Написали второй код для второй программы (рис. 3.4).

Рис. 3.4: второй код для второй программы

Проверили код на работу (рис. 3.5).

Рис. 3.5: Проверили код на работу

Написали код для третьей программы (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.))(рис. 3.6).

```

1 #!/bin/bash
2
3 create_files() {
4     local count=$1
5     for ((i=1; i<=count; i++)); do
6         touch "$i.tmp"
7         echo "Создан файл $i.tmp"
8     done
9 }
10
11 delete_files() {
12     local count=$1
13     for ((i=1; i<=count; i++)); do
14         if [ -e "$i.tmp" ]; then
15             rm "$i.tmp"
16             echo "Удален файл $i.tmp"
17         fi
18     done
19 }
20
21 if [ $# -eq 0 ]; then
22     echo "Не указано количество файлов для создания"
23     exit 1
24 fi
25
26 action=$1
27
28 case $action in
29     create)
30         create_files $2
31         ;;
32     delete)
33         delete_files $2

```

Рис. 3.6: код для третьей программы

Проверили код на работу (рис. 3.7).

```

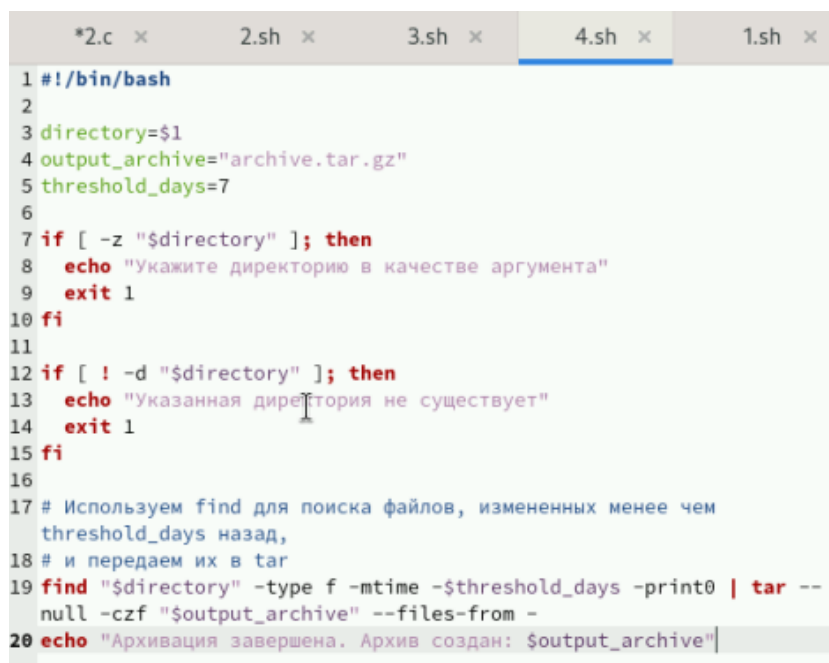
[ksyusha@ksyusha lab13]$ ./3.sh create 3
Создан файл 1.tmp
Создан файл 2.tmp
Создан файл 3.tmp
[ksyusha@ksyusha lab13]$ ./3.sh delete 3
Удален файл 1.tmp
Удален файл 2.tmp
Удален файл 3.tmp

```

Рис. 3.7: Проверили код на работу

Написали код для четвертой программы (Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории.)

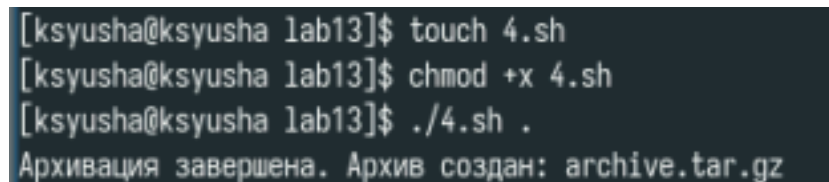
(рис. 3.8).



```
*2.c x 2.sh x 3.sh x 4.sh x 1.sh x
1 #!/bin/bash
2
3 directory=$1
4 output_archive="archive.tar.gz"
5 threshold_days=7
6
7 if [ -z "$directory" ]; then
8     echo "Укажите директорию в качестве аргумента"
9     exit 1
10 fi
11
12 if [ ! -d "$directory" ]; then
13     echo "Указанная директория не существует"
14     exit 1
15 fi
16
17 # Используем find для поиска файлов, измененных менее чем
18 #   threshold_days назад,
19 # и передаем их в tar
20 find "$directory" -type f -mtime -$threshold_days -print0 | tar --
    null -czf "$output_archive" --files-from -
20 echo "Архивация завершена. Архив создан: $output_archive"
```

Рис. 3.8: код для четвертой программы

Проверили код на работу (рис. 3.9).



```
[ksyusha@ksyusha lab13]$ touch 4.sh
[ksyusha@ksyusha lab13]$ chmod +x 4.sh
[ksyusha@ksyusha lab13]$ ./4.sh .
Архивация завершена. Архив создан: archive.tar.gz
```

Рис. 3.9: Проверили код на работу

4 Ответы на контрольные вопросы

1. Каково предназначение команды getopts?

Команда `getopts` используется в `shell`-скриптах для обработки опций и аргументов, переданных скрипту через командную строку. Она позволяет скрипту получать опции, а также их соответствующие аргументы, если они предусмотрены. Это упрощает создание скриптов, которые могут гибко настраиваться в зависимости от переданных параметров.

2. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы (или подстановочные символы) в `shell` (например, `*`, `?`, `[]`) используются для генерации списков файлов, соответствующих определённому шаблону. Например:

- `*` (звёздочка) - соответствует любому количеству символов (включая отсутствие символов). Например, `*.txt` выберет все файлы с расширением `".txt"`.
- `?` (вопросительный знак) - соответствует одному любому символу. Например, `file?.txt` выберет `file1.txt`, `fileA.txt` и т.д.
- `[]` (квадратные скобки) - соответствует одному символу из указанного набора. Например, `file[1-3].txt` выберет `file1.txt`, `file2.txt` и `file3.txt`.

Когда вы используете метасимволы в командах, `shell` перед выполнением команды разворачивает

3. Какие операторы управления действиями вы знаете?

В bash (и других shell) есть следующие операторы управления действиями (или потоком управления):

- `if...then...elif...else...fi`: Условный оператор, позволяющий выполнять различные блоки кода в зависимости от выполнения заданных условий.
- `case...esac`: Оператор выбора, позволяющий выбирать один из нескольких вариантов выполнения кода, основываясь на значении переменной.
- `for...do...done`: Цикл, позволяющий повторять блок кода для каждого элемента в списке.
- `while...do...done`: Цикл, выполняющийся до тех пор, пока заданное условие истинно.
- `until...do...done`: Цикл, выполняющийся до тех пор, пока заданное условие ложно.
- `break`: Оператор, позволяющий выйти из цикла до его завершения.
- `continue`: Оператор, позволяющий перейти к следующей итерации цикла, пропуская оставшуюся часть текущей итерации.

4. Какие операторы используются для прерывания цикла?

Как уже упоминалось выше, для прерывания цикла используются операторы `break` (для полного выхода из цикла) и `continue` (для перехода к следующей итерации цикла).

5. Для чего нужны команды `false` и `true`?

Команды `false` и `true` используются для возврата логических значений “ложь” (с кодом выхода 1) и “истина” (с кодом выхода 0), соответственно. Они часто используются в условных выражениях (например, в операторах `if`, `while`, `until`), где необходимо явное указание логического значения.

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

Эта строка означает следующее:

- `if test -f man/i.s` : , *man/i.s* обычным существующим файлом. Команда `test` (или эквивалентная ей `[]`) используется для проверки различных условий. `-f` - это опция `test`, которая проверяет, существует ли файл и является ли он обычным файлом (не директорией, символической ссылкой и т.д.).

- `man$`: Предположительно, это часть пути к директории.

- `$i`: Это переменная, значение которой будет подставлено. Скорее всего, это переменная цикла.

- `$s`: Это другая переменная, значение которой тоже будет подставлено. Обычно используется как расширение файла.

В итоге, строка проверяет существование файла, имя которого получается путем объединения строки `"man$"`, значения переменной `$i`, символа `"."`, и значения переменной `$s`.

7. Объясните различия между конструкциями `while` и `until`.

- `while`: Цикл `while` выполняет блок кода до тех пор, пока условие истинно. Как только условие становится ложным, цикл прекращается.

`until`: Цикл `until` выполняет блок кода до тех пор, пока условие ложно*. Как только условие становится истинным, цикл прекращается.

Основное отличие: `while` выполняет, пока условие истинно, а `until` выполняет, пока условие ложно. Они логически противоположны друг другу.

5 Выводы

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