

Лабораторная работа №12 по предмету
Операционные системы

Группа НПМбв-01-19

Бондаренко Артем Федорович

Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	8
Выводы	18
Ответы на контрольные вопросы	19

Список иллюстраций

1	Код программы	9
2	Файл input	10
3	Запуск созданного командного файла с соответствующими аргументами и опциями	10
4	Созданный в результате исполнения программы файл output с искомыми строками	11
5	Код программы написанный на Си	12
6	Код программы написанный на bash	13
7	Результат выполнения командного файла в терминале	13
8	Код программы	14
9	Результат выполнения командного файла в терминале без аргумента delete, демонстрация созданных файлов	15
10	Результат выполнения командного файла в терминале со вторым аргументом delete, демонстрация созданных файлов	15
11	Код программы	16
12	Результат выполнения командного файла в терминале	17

Список таблиц

Цель работы

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

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 `-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 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 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`).

Выполнение лабораторной работы

Используя команды `getopts` `grep`, написал командный файл, который анализирует командную строку с ключами:

- `-input_file` — прочитать данные из указанного файла;
- `-output_file` — вывести данные в указанный файл;
- `-pattern` — указать шаблон для поиска;
- `-case_sensitive` — различать большие и малые буквы;
- `-line-number` — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом `-p`. (Ссылка: Рис.1) (Ссылка: Рис.3)


```

input_file=""
output_file=""
pattern=""
case_sensitive=""
line_numbers=""

# Анализируем командную строку с помощью команды getopt
while getopt "i:o:p:Cn" opt; do
    case $opt in
        i)
            input_file=$OPTARG
            ;;
        o)
            output_file=$OPTARG
            ;;
        p)
            pattern=$OPTARG
            ;;
        C)
            case_sensitive="-i"
            ;;
        n)
            line_numbers="-n"
            ;;
        \?)
            echo "Invalid option: -$OPTARG" >&2
            exit 1
            ;;
    esac
done

# Используем grep для поиска строк в файле
grep $case_sensitive $line_numbers "$pattern" "$input_file"

# Если указан файл для вывода, перенаправляем вывод в него
if [ -n "$output_file" ]; then
    grep $case_sensitive $line_numbers "$pattern" "$input_file" > "$output_file"
fi

```

Рис. 1: Код программы

Результат работы командного файла (Ссылка: Рис.2)

Тестовый файл из которого командный файл будет искать символы (Ссылка: Рис.2)

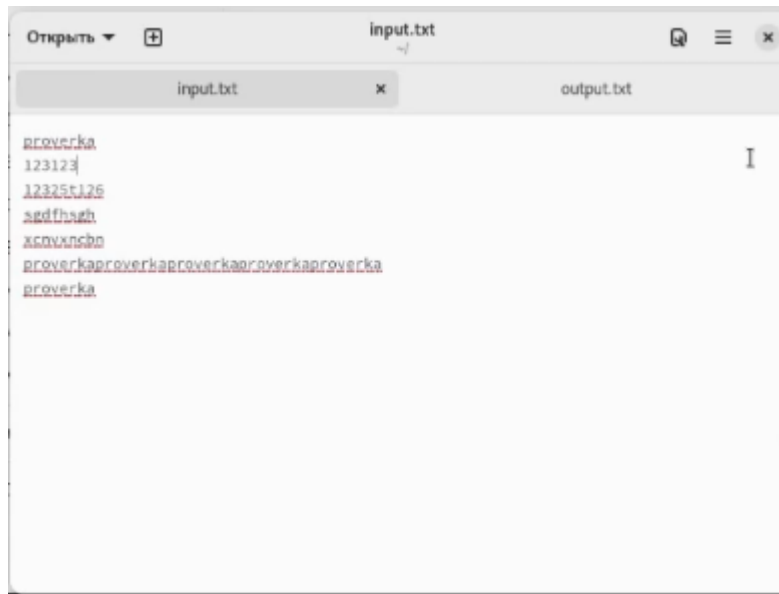


Рис. 2: Файл input



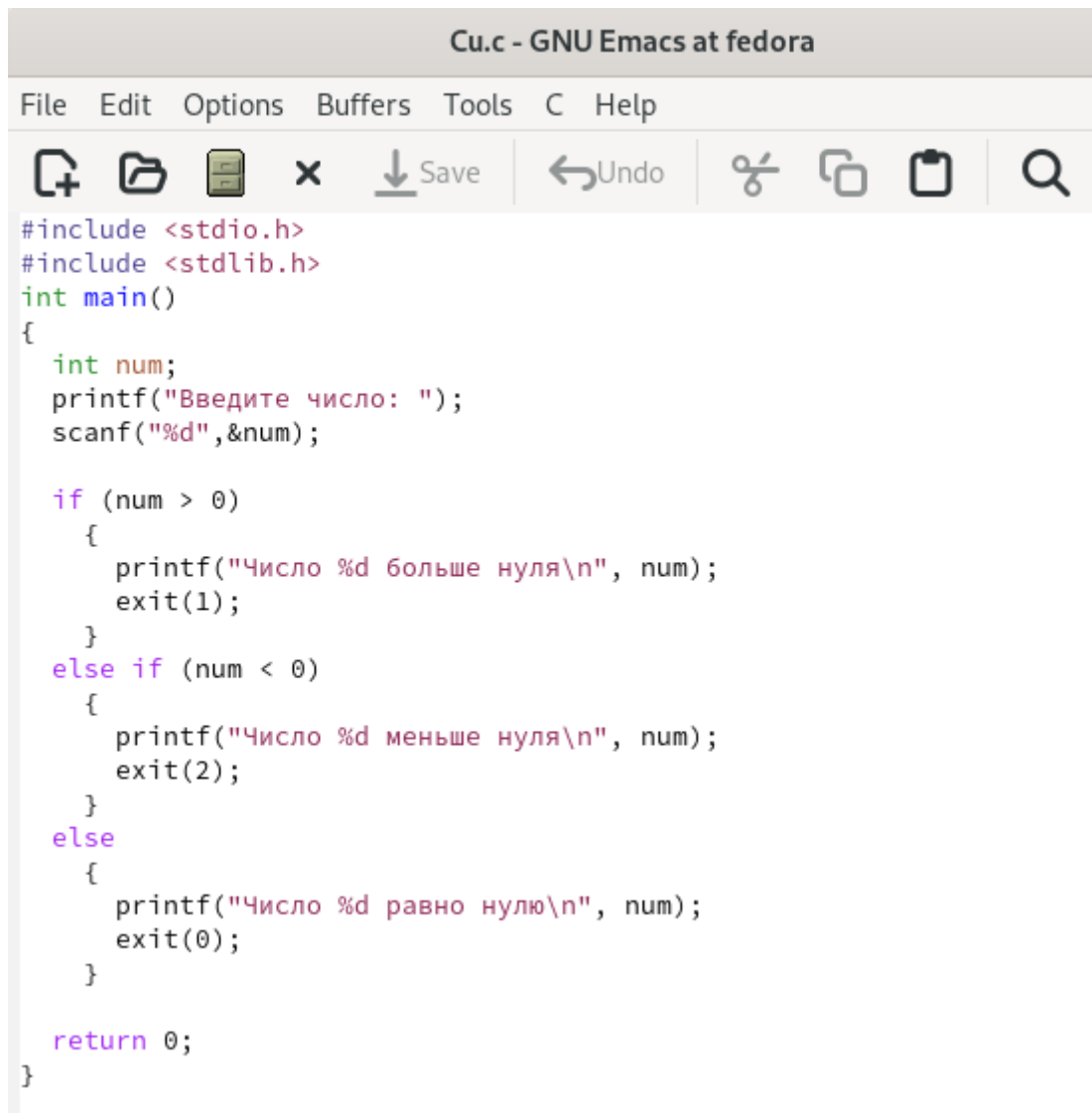
Рис. 3: Запуск созданного командного файла с соответствующими аргументами и опциями

После исполнения командного файла искомые результаты сохраняются в отдельном файле output (Ссылка: Рис.4)



Рис. 4: Созданный в результате исполнения программы файл output с искомыми строками

Написал на языке Си программу, в которую нужно ввести число, после чего она определяет, является ли введенное число больше нуля, меньше нуля или равно нулю. (Ссылка: Рис.5)



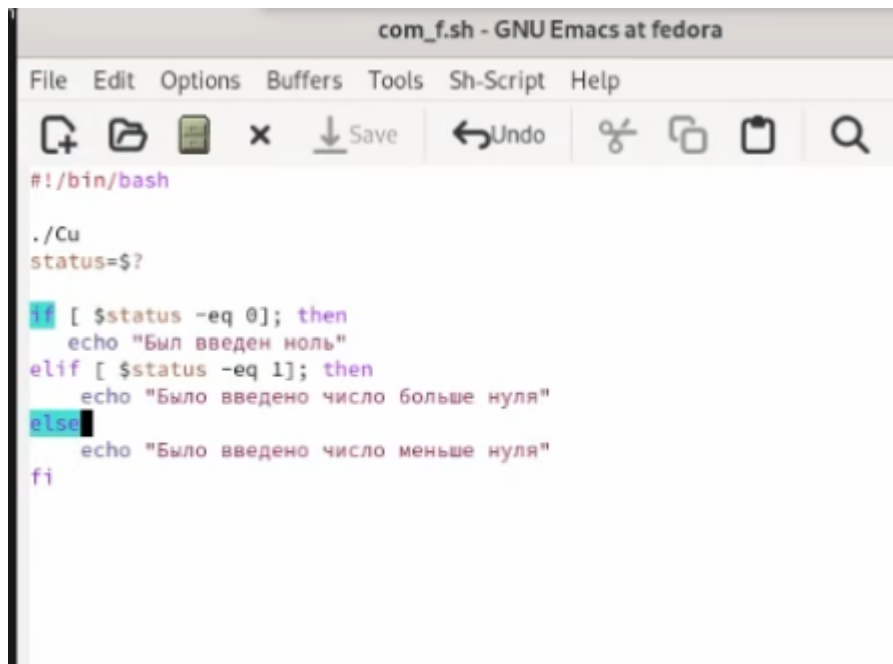
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num;
    printf("Введите число: ");
    scanf("%d",&num);

    if (num > 0)
    {
        printf("Число %d больше нуля\n", num);
        exit(1);
    }
    else if (num < 0)
    {
        printf("Число %d меньше нуля\n", num);
        exit(2);
    }
    else
    {
        printf("Число %d равно нулю\n", num);
        exit(0);
    }

    return 0;
}
```

Рис. 5: Код программы написанный на Си

Затем с помощью команды `gcc` скомпилировал её в исполняемый файл и далее написал командный файл на `bash` (Ссылка: Рис.6), который вызывает программу на Си и, проанализировав её с помощью команды `$?`, выдает сообщение о том, какое число было введено. (Ссылка: Рис.7)

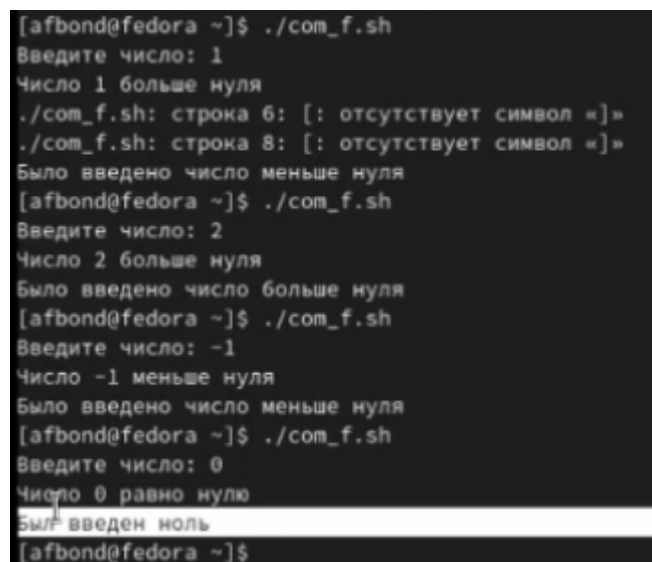


```
#!/bin/bash

./Cu
status=$?

if [ $status -eq 0 ]; then
    echo "Был введен ноль"
elif [ $status -eq 1 ]; then
    echo "Было введено число больше нуля"
else
    echo "Было введено число меньше нуля"
fi
```

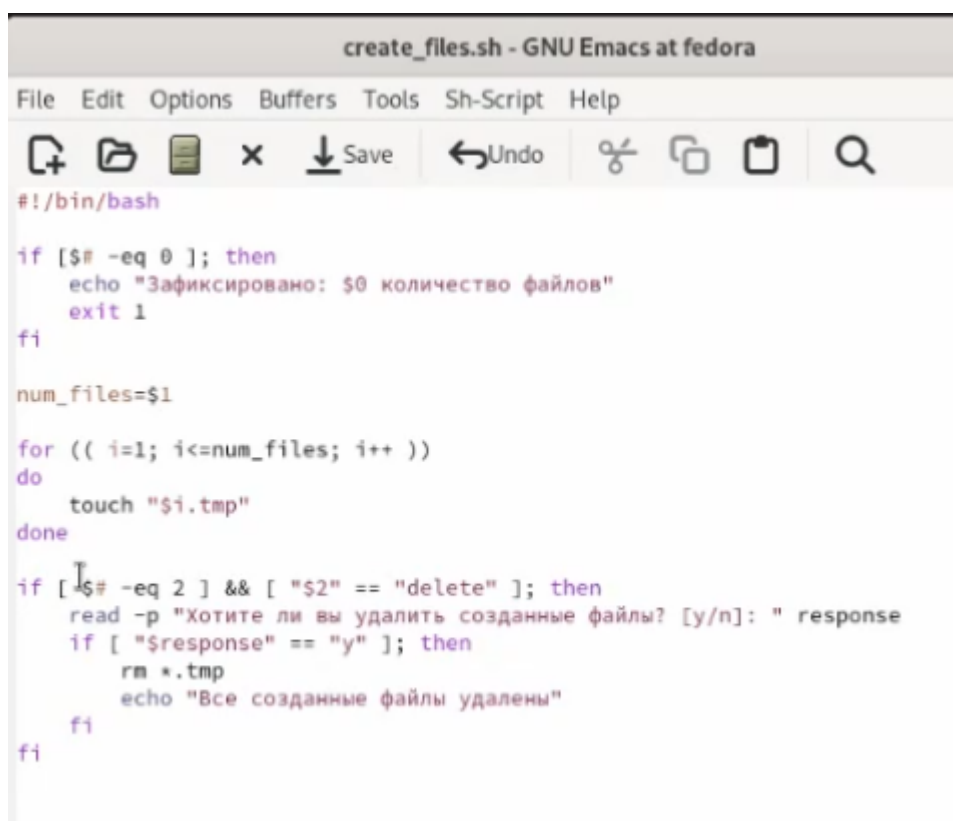
Рис. 6: Код программы написанный на bash



```
[afbond@fedora ~]$ ./com_f.sh
Введите число: 1
Число 1 больше нуля
./com_f.sh: строка 6: [: отсутствует символ «]»
./com_f.sh: строка 8: [: отсутствует символ «]»
Было введено число меньше нуля
[afbond@fedora ~]$ ./com_f.sh
Введите число: 2
Число 2 больше нуля
Было введено число больше нуля
[afbond@fedora ~]$ ./com_f.sh
Введите число: -1
Число -1 меньше нуля
Было введено число меньше нуля
[afbond@fedora ~]$ ./com_f.sh
Введите число: 0
Число 0 равно нулю
Был введен ноль
[afbond@fedora ~]$
```

Рис. 7: Результат выполнения командного файла в терминале

Написал командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до i (например 1.tmp, 2.tmp, 3.tmp,4.tmp и т.д.). (Ссылка: Рис.8)



```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Зафиксировано: $0 количество файлов"
    exit 1
fi

num_files=$1

for (( i=1; i<=num_files; i++ ))
do
    touch "$i.tmp"
done

if [ $# -eq 2 ] && [ "$2" == "delete" ]; then
    read -p "Хотите ли вы удалить созданные файлы? [y/n]: " response
    if [ "$response" == "y" ]; then
        rm *.tmp
        echo "Все созданные файлы удалены"
    fi
fi
```

Рис. 8: Код программы

Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот командный файл умеет удалять созданные файлы, был написан код который позволяет задать второй аргумент “delete”, после чего пользователю будет предложен выбор, удалять созданные файлы или нет. (Ссылка: Рис.9) (Ссылка: Рис.10)

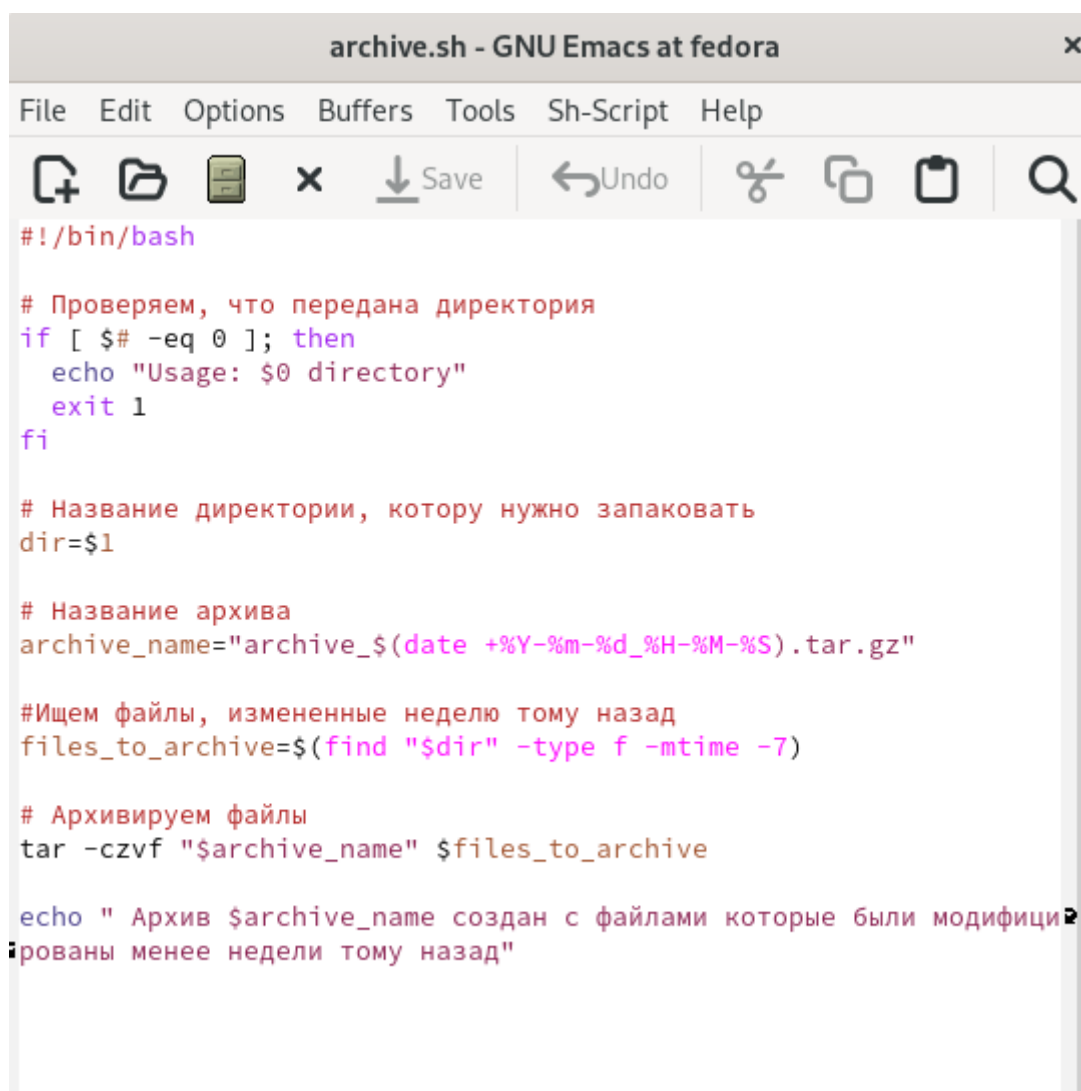
```
[afbond@fedora ~]$ ./create_files.sh 2
[afbond@fedora ~]$ ls
1.tmp          Cu
2.tmp          Cu.c
backup_script.sh  Cu.c~
com_f.sh~      file.txt
conf.txt       input.txt
count.sh       install-tl-20230326
count.sh~      L10
create_files.sh L11
create_files.sh~ L12_1-
[afbond@fedora ~]$
```

Рис. 9: Результат выполнения командного файла в терминале без аргумента delete, демонстрация созданных файлов

```
[afbond@fedora ~]$ ./create_files.sh 6 delete
Хотите ли вы удалить созданные файлы? [y/n]: y
Все созданные файлы удалены
[afbond@fedora ~]$ ls
backup_script.sh  Cu.c~      L6
com_f.sh~        file.txt   L7
conf.txt         input.txt  L8
count.sh         install-tl-20230326 L9
count.sh~        L10       lab07
create_files.sh  L11       lab07
create_files.sh~ L12_1-    LOL..
Cu               L12_1.sh  '#my_
Cu.c             L12_1.sh~ my_sc
```

Рис. 10: Результат выполнения командного файла в терминале со вторым аргументом delete, демонстрация созданных файлов

Написал командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировал его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад. (Ссылка: Рис.11) (Ссылка: Рис.12)



```
#!/bin/bash

# Проверяем, что передана директория
if [ $# -eq 0 ]; then
    echo "Usage: $0 directory"
    exit 1
fi

# Название директории, которую нужно запаковать
dir=$1

# Название архива
archive_name="archive_$(date +%Y-%m-%d_%H-%M-%S).tar.gz"

# Ищем файлы, измененные неделю тому назад
files_to_archive=$(find "$dir" -type f -mtime -7)

# Архивируем файлы
tar -czvf "$archive_name" $files_to_archive

echo " Архив $archive_name создан с файлами которые были модифицированы менее недели тому назад"
```

Рис. 11: Код программы

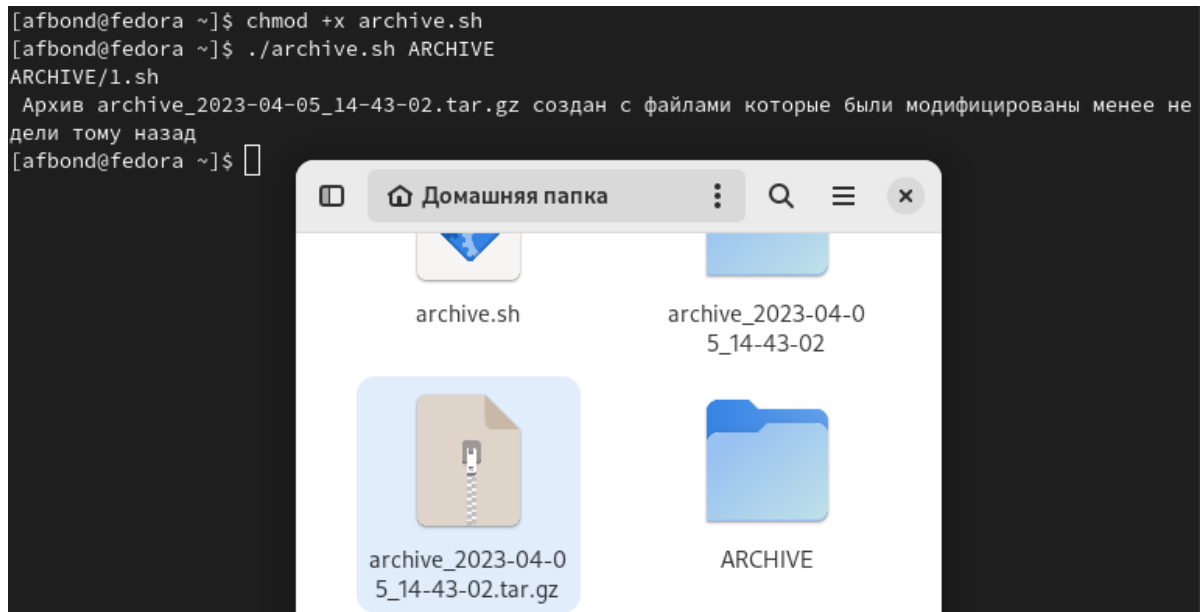


Рис. 12: Результат выполнения командного файла в терминале

Выводы

Таким образом, мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов. Например, командные файлы которые анализируют командную строку с ключами на языке Си и которые выводят необходимый запрос, научились создавать командные файлы которые в автоматическом режиме запаковывают в архив все необходимые файлы.

Ответы на контрольные вопросы

1. Каково предназначение команды `getopts`?

Команда `getopts` в скриптах оболочки Unix/Linux предназначена для обработки опций командной строки. Она позволяет скрипту получать аргументы командной строки в определенном формате, что делает его более удобным для использования конечными пользователями.

Команда `getopts` используется в цикле `while`, который обрабатывает каждый аргумент командной строки по очереди. При этом можно задать различные флаги и параметры для каждой опции, чтобы указать, как они должны быть обработаны. Например, вы можете указать, что опция должна иметь параметр, или что она является флагом, который просто включается или выключается.

Кроме того, `getopts` позволяет обрабатывать несколько опций в одном аргументе командной строки, а также определить опцию по умолчанию, которая будет использоваться, если пользователь не указал никаких опций.

2. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы в Unix-подобных операционных системах используются для генерации имён файлов с помощью шаблонов. Эти метасимволы, такие как звездочка (*), вопросительный знак (?) и квадратные скобки ([]), могут использоваться для указания шаблона имени файла, который должен соответствовать определенному шаблону.

Например, вы можете использовать метасимвол звездочки для указания шаблона имени файла, который должен начинаться с "log" и заканчиваться на любое расши-

рение файла, например, “.txt” или “.log”. Вы можете использовать команду ls для вывода всех файлов, соответствующих этому шаблону, следующим образом:

```
ls log*
```

Это выведет все файлы, начинающиеся с “log” и имеющие любое расширение файла, находящиеся в текущей директории.

Кроме того, метасимволы также могут использоваться для генерации имён файлов с помощью команд, таких как cp, mv и rm. Например, вы можете использовать метасимвол звездочки для копирования всех файлов, начинающихся с “log” в другую директорию следующим образом:

```
cp log* /path/to/new/directory/
```

Это скопирует все файлы, начинающиеся с “log”, в директорию /path/to/new/directory/.

3. Какие операторы управления действиями вы знаете?

В языке программирования bash есть много операторов управления действиями, вот некоторые из них:

if-then-else: оператор условия, позволяет выполнять определенные действия в зависимости от условия.

for: оператор цикла, позволяет выполнить набор инструкций несколько раз, итерируясь по списку значений.

while: оператор цикла, позволяет выполнять набор инструкций, пока определенное условие остается истинным.

case: оператор выбора, позволяет выбрать действие, которое должно быть выполнено, исходя из значения переменной.

until: оператор цикла, выполняет набор инструкций до тех пор, пока определенное условие не станет истинным.

select: оператор выбора, позволяет создать меню выбора для пользователя.

break: оператор, который прерывает выполнение цикла или выбора.

`continue`: оператор, который пропускает текущую итерацию цикла или выбора и переходит к следующей.

`trap`: оператор, который позволяет устанавливать обработчики сигналов и выполнять определенные действия при получении сигнала.

`shift`: оператор, который позволяет сдвигать значения параметров командной строки на одну позицию.

Это только некоторые из операторов управления действиями в языке `bash`.

4. Какие операторы используются для прерывания цикла?

В языке программирования `bash` используется оператор `break` для немедленного выхода из цикла и оператор `continue` для перехода к следующей итерации цикла, игнорируя все оставшиеся команды в текущей итерации. Оба оператора могут использоваться в циклах `for`, `while` и `until`.

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

Команды `false` и `true` - это утилиты командной строки в `Linux` и других системах `Unix`, которые не выполняют никаких действий, кроме возврата кода возврата 1 или 0 соответственно.

Команда `false` всегда завершается неудачно (возвращает код возврата 1), поэтому ее можно использовать в качестве фиктивной команды, когда необходимо имитировать ошибку или неправильное завершение команды.

Команда `true`, наоборот, всегда завершается успешно (возвращает код возврата 0), и ее можно использовать для имитации успешного выполнения команды.

Обе команды могут использоваться в сценариях командной строки для тестирования или отладки скриптов, а также в качестве заполнителей в пайпах или последовательностях команд, когда нужно предотвратить ошибки, связанные с незавершением команд.

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

Строка `if test -f mans/i.s` проверяет наличие файла `i.s` в каталоге `mans`.

Более подробно:

`if` - ключевое слово, указывающее на начало условной конструкции.

`test` - команда, используемая для проверки различных условий в shell-скриптах. В данном случае она используется для проверки существования файла.

`-f` - параметр `test`, указывающий, что мы проверяем существование обычного файла.

`mans` - имя каталога, где ожидается нахождение файла, с переменной `$s`, которая вероятно задаёт расширение файла.

`i` - имя файла, которое мы проверяем.

`$s` - переменная, которая вероятно задаёт расширение файла.

Таким образом, эта строка проверяет наличие файла с именем `i.s` в каталоге `mans`, где `$s` и `$i` заменены на соответствующие значения. Если файл существует, то условие возвращает значение `true`, и блок кода внутри условной конструкции выполняется, а если файл не существует, то условие возвращает значение `false`, и блок кода внутри условной конструкции не выполняется.

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

Конструкция `while` используется для выполнения цикла, пока условие истинно. Цикл выполняется, пока результат выражения внутри скобок `while` истинен.

Конструкция `until` похожа на `while`, но выполняет цикл, пока условие ложно. Цикл будет выполняться, пока результат выражения внутри скобок `until` ложен.