

# **Отчёта по лабораторной работе №11**

**Лабораторная работа №11. Программирование в командном  
процессоре ОС UNIX. Ветвления и циклы**

Надежда Александровна Рогожина

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
3.1	Метасимволы . . . . .	8
3.2	Использование команды getopts . . . . .	9
3.3	Управление последовательностью действий в командных файлах	9
3.4	Оператор выбора case . . . . .	10
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>12</b>
<b>5</b>	<b>Выводы</b>	<b>19</b>
	<b>Список литературы</b>	<b>20</b>

## Список иллюстраций

4.1	Скрипт №1 . . . . .	12
4.2	Запуск первого скрипта . . . . .	13
4.3	Результат . . . . .	13
4.4	Программа на С . . . . .	14
4.5	Доустановка . . . . .	14
4.6	Компиляция . . . . .	14
4.7	Скрипт №2 . . . . .	15
4.8	Выполнение скрипта №2 . . . . .	15
4.9	Скрипт №3 . . . . .	16
4.10	Выполнение скрипта №3 . . . . .	16
4.11	Повторное выполнение скрипта №3 . . . . .	16
4.12	Скрипт №4 . . . . .	17
4.13	Запуск скрипта . . . . .	17
4.14	Папка out . . . . .	18

## 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;
- `-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 `o` 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  $\infty$  (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 Теоретическое введение

### 3.1 Метасимволы

При перечислении имён файлов текущего каталога можно использовать следующие символы:

- `*` — соответствует произвольной, в том числе и пустой строке;
- `?` — соответствует любому одинарному символу;
- `[c1-c1]` — соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`.

Например, `echo *` — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls .c` — выведет все файлы с последними двумя символами, совпадающими с `.c`. `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.

Такие символы, как `' < > * ? | \ " &`, являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом.



## 3.2 Использование 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, для команды `ls` флагом может являться `-F`. Иногда флаги имеют аргументы, связанные с ними. Программы интерпретируют флаги, соответствующим образом изменяя своё поведение.

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 `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введённые данные с помощью оператора `case`.

## 3.3 Управление последовательностью действий в командных файлах

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 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. Так например, команда

```
test -f file
```

возвращает нулевой код завершения (истина), если файл `file` существует, и ненулевой код завершения (ложь) в противном случае:

- `test s` — истина, если аргумент `s` имеет значение истина;
- `test -f file` — истина, если файл `file` существует;
- `test -i file` — истина, если файл `file` доступен по чтению;
- `test -w file` — истина, если файл `file` доступен по записи;
- `test -e file` — истина, если файл `file` — исполняемая программа;
- `test -d file` — истина, если файл `file` является каталогом

### 3.4 Оператор выбора `case`

Оператор выбора `case` реализует возможность ветвления на произвольное число ветвей. Эта возможность обеспечивается в большинстве современных языков программирования, предполагающих использование структурного подхода.

В обобщённой форме оператор выбора `case` выглядит следующим образом:

```
case имя in  
шаблон1) список-команд;;
```

```
шаблон2) список-команд;;
```

```
...
```

```
esac
```

Выполнение оператора выбора case сводится к тому, что выполняется последовательность команд (операторов), задаваемая списком список-команд, в строке, для которой значение переменной имя совпадает с шаблоном. Поскольку метасимвол \* соответствует произвольной, в том числе и пустой, последовательности символов, то его можно использовать в качестве шаблона в последней строке перед служебным словом esac. В этом случае реализуются все действия, которые необходимо произвести, если значение переменной имя не совпадает ни с одним из шаблонов, заданных в предшествующих строках.

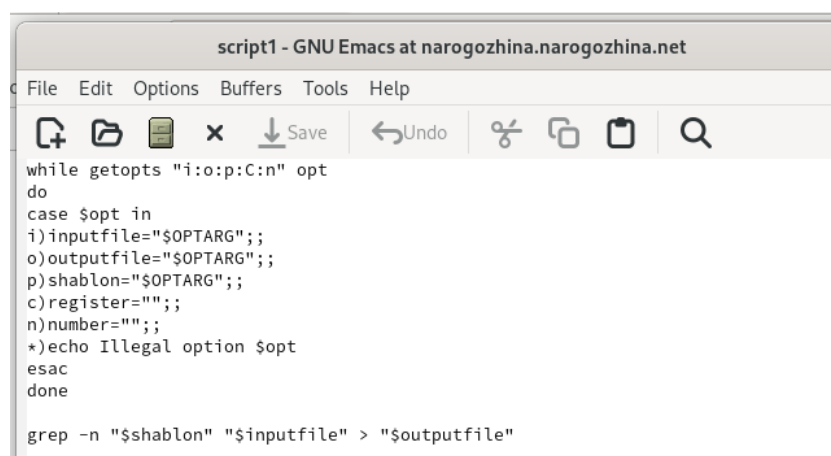
## 4 Выполнение лабораторной работы

Для выполнения лабораторной работы будем пользоваться текстовым редактором Emacs.

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` (рис. 4.1,4.2,4.3):



```
script1 - GNU Emacs at narogozhina.narogozhina.net
File Edit Options Buffers Tools Help
while getopts "i:o:p:C:n" opt
do
case $opt in
i)inputfile="$OPTARG";;
o)outputfile="$OPTARG";;
p)shablon="$OPTARG";;
c)register="";;
n)number="";;
*)echo Illegal option $opt
esac
done
grep -n "$shablon" "$inputfile" > "$outputfile"
```

Рис. 4.1: Скрипт №1

После первого запуска, я подправила текст скрипта (готовый вариант прикрепила выше) и запустила снова.

```
[narogozhina@narogozhina ~]$ emacs script1
[narogozhina@narogozhina ~]$ chmod +x script1
[narogozhina@narogozhina ~]$ ./script1 -i conf.txt -o output.txt -pm etconf -C -n
./script1: строка 1: getops: команда не найдена
./script1: строка 13: : Нет такого файла или каталога
[narogozhina@narogozhina ~]$ emacs script1
[narogozhina@narogozhina ~]$ ./script1 -i conf.txt -o output.txt -pm etconf -C -n
[narogozhina@narogozhina ~]$ emacs script1
[narogozhina@narogozhina ~]$ ./script1 -i conf.txt -o output.txt -pm etconf -C -n
```

Рис. 4.2: Запуск первого скрипта

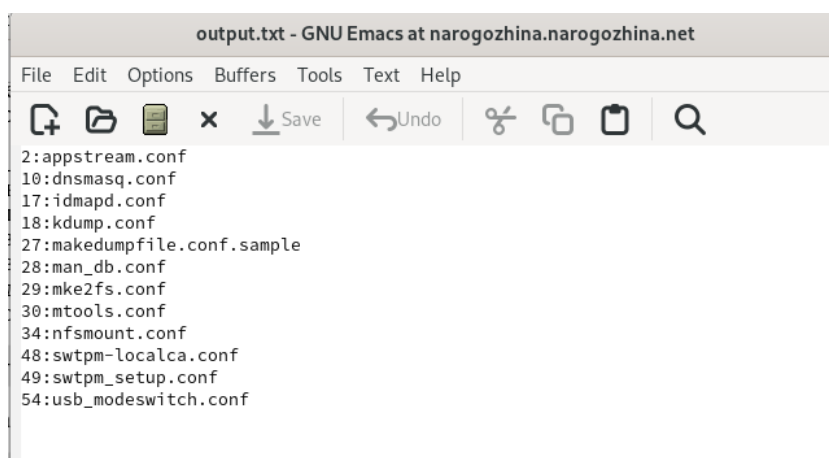


Рис. 4.3: Результат

В данном случае, результат выполнения скрипта сохранился в текстовый файл output.txt

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 (рис. 4.4):

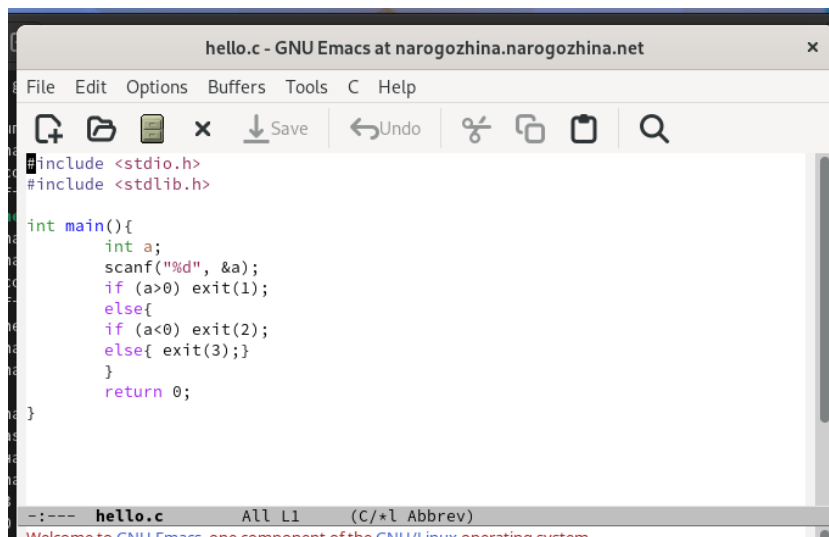


Рис. 4.4: Программа на C

Чтобы программа работала нормально, пришлось доустановить gcc компилятор с++ (рис. 4.5):

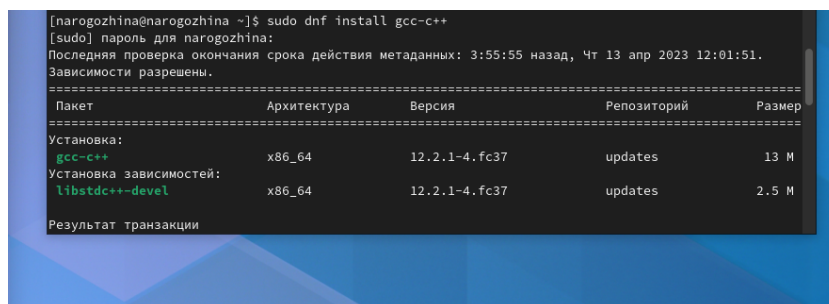


Рис. 4.5: Доустановка

Далее скомпилируем файл (рис. 4.6):

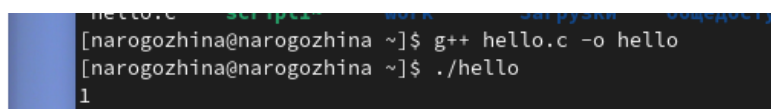


Рис. 4.6: Компиляция

Сам скрипт (рис. 4.7):

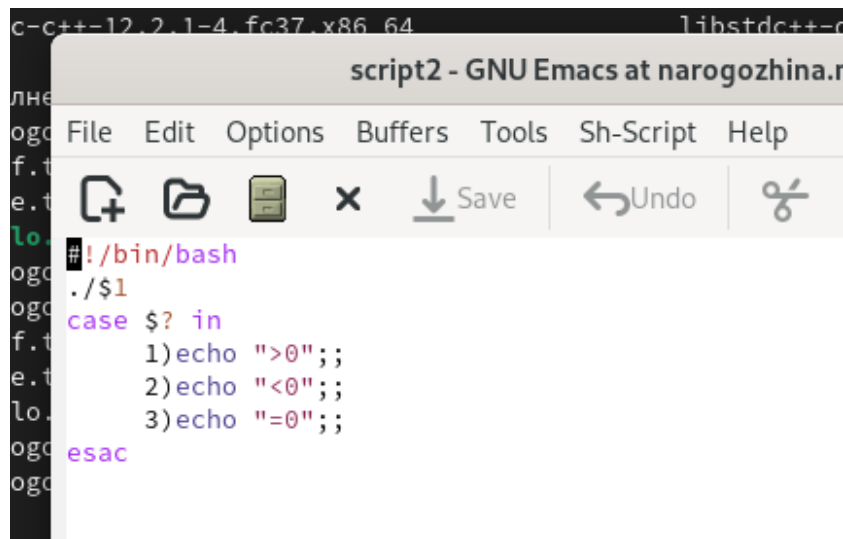


Рис. 4.7: Скрипт №2

Пример выполнения (рис. 4.8):

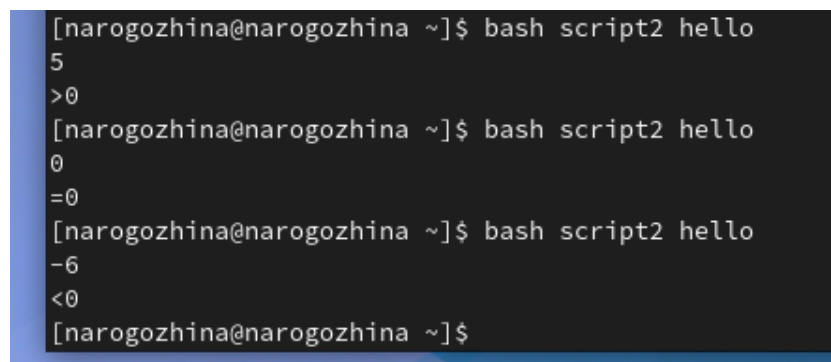


Рис. 4.8: Выполнение скрипта №2

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) (рис. 4.9):

```
script3 - GNU Emacs at narogozhina.narogozhina.net
File Edit Options Buffers Tools Sh-Script Help
[Icons: Copy, Paste, Save, Undo, Redo, Find, etc.]
#!/bin/bash
N=$1+1
i=1
until $(let "$N"=="$i")
do
    if test -f "$i".tmp
    then rm "$i".tmp
    else touch "$i".tmp
    fi
    let i+=1
done
```

Рис. 4.9: Скрипт №3

Пример выполнения (первый раз) (рис. 4.10):

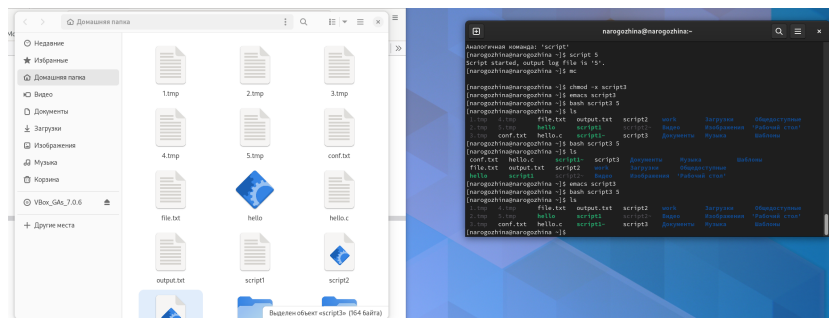


Рис. 4.10: Выполнение скрипта №3

Выполнение повторно (рис. 4.11):

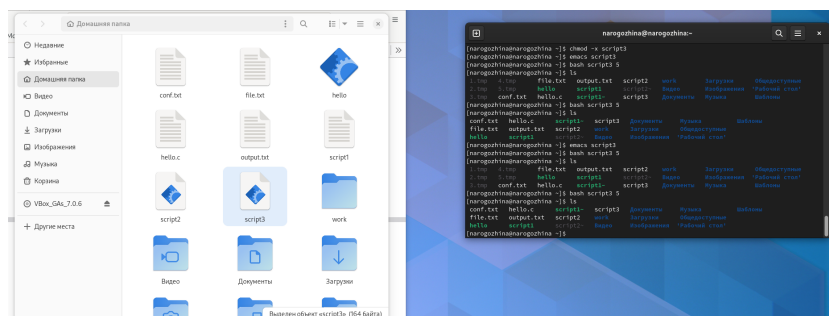
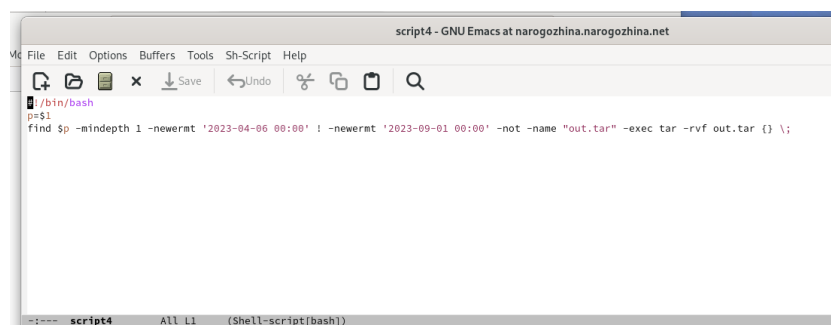


Рис. 4.11: Повторное выполнение скрипта №3



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) (рис. 4.12):



```
script4 - GNU Emacs at narogozhina.narogozhina.net
File Edit Options Buffers Tools Sh-Script Help
p=$1
find $p -mindepth 1 -newermt '2023-04-06 00:00' ! -newermt '2023-09-01 00:00' -not -name "out.tar" -exec tar -rvf out.tar {} \;
```

Рис. 4.12: Скрипт №4

Пример выполнения (рис. 4.13):

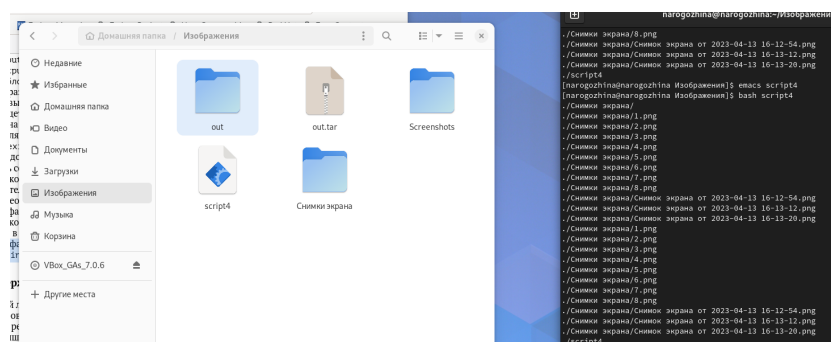


Рис. 4.13: Запуск скрипта

Здесь представлен архив out, который был создан во время работы скрипта, и в нем (рис. 4.14):

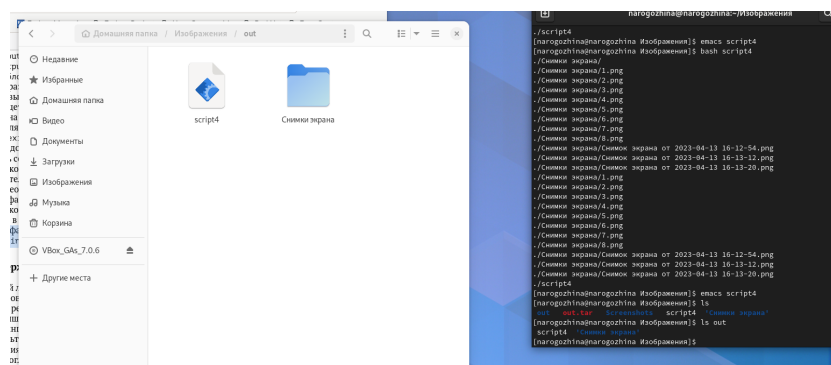


Рис. 4.14: Папка out

Я запускала этот скрипт именно в папке “Изображения”, т.к. в домашнем каталоге слишком много недавно изменённых файлов.

## 5 Выводы

В ходе лабораторной работы мы научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## **Список литературы**

1. Руководство по выполнению лабораторной работы №10
2. Руководство по выполнению лабораторной работы №11