

# **Лабораторная работа 11**

**Программирование в командном процессоре ОС UNIX. Ветвления и  
циклы**

Генералов Даниил, НПИ-01-21, 1032212280

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	11
5	Выводы	13

## Список иллюстраций

3.1	Программа поиска . . . . .	7
3.2	Результат работы программы поиска . . . . .	7
3.3	Программа сравнения чисел . . . . .	8
3.4	Результат работы программы сравнения чисел . . . . .	8
3.5	Программа создания файлов . . . . .	9
3.6	Результат работы программы создания файлов . . . . .	9
3.7	Программа архивирования . . . . .	10
3.8	Результат работы программы архивирования . . . . .	10

## Список таблиц

# 1 Цель работы

Целью данной работы является:

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

Требуется написать 4 командных файла:

- принимает аргументы с командной строки и использует их для запуска `grep`
- запускает другую программу и получает ее код выхода
- создает файлы с названиями, имеющими номер в последовательности
- создает архив из файлов, которые обновлены меньше чем неделю назад

## 2 Теоретическое введение

Командный процессор (*shell*) – это программа на Unix-системах, которая принимает ввод от пользователя и исполняет инструкции. Помимо интерактивного использования, она может исполнять список команд, заданный в файле, и она обладает набором команд, достаточным для написания программ разной степени сложности. В этой работе мы продолжаем рассматривать этот функционал командного процессора, составляя несколько командных файлов, выполняющих определенные действия.

## 3 Выполнение лабораторной работы

Первая программа представлена на рис. 3.1. Она принимает несколько аргументов с помощью `getopts` и запоминает их значения. После этого, если все значения правильные, они подставляются в команду `grep`, которая читает один файл, ищет там строки по заданному шаблону и выводит их в файл. Это показано на рис. 3.2.

```
1 #!/bin/bash
2
3 while getopts "i:o:p:Cn" opt; do
4     case $opt in
5         i)
6             input_file=$OPTARG
7             ;;
8         o)
9             output_file=$OPTARG
10            ;;
11         p)
12             pattern=$OPTARG
13            ;;
14         C)
15             case_insensitive=1
16            ;;
17         n)
18             print_line_numbers=1
19            ;;
20         *)
21             echo "Invalid option: -$OPTARG" >&2
22             exit 1
23            ;;
24     esac
25     echo "Option -$OPTARG requires an argument." >&2
26     exit 1
27 done
28
29 if [ -z "$input_file" ] || [ -z "$output_file" ] || [ -z "$pattern" ]; then
30     echo "Usage: $0 -i <input_file> -o <output_file> -p <pattern> [-C] [-n]"
31     exit 1
32 fi
33
34 grep $(if [ -n "$case_insensitive" ]; then echo "-i"; fi) $(if [ -n "$print_line_numbers" ]; then printf "%s -n"; fi) "$pattern" "$input_file" > "$output_file"
```

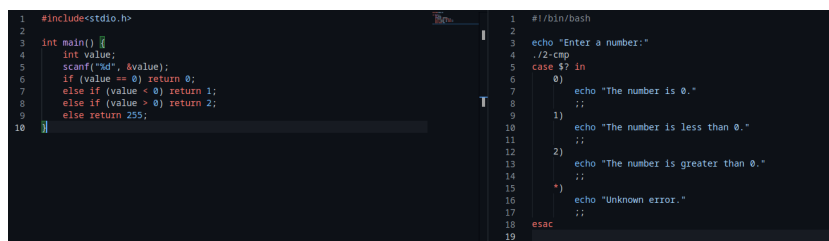
Рис. 3.1: Программа поиска

```
[danyaidanya-GE72NVR-78G code]$ ./l-grep-frontend.sh
Usage: ./l-grep-frontend.sh -i <input_file> -o <output_file> -p <pattern> [-C] [-n]
[danyaidanya-GE72NVR-78G code]$ ./l-grep-frontend.sh -i l-grep-frontend.sh -o test -p grep
[danyaidanya-GE72NVR-78G code]$ cat test
grep $(if [ -n "$case_insensitive" ]; then echo "-i"; fi) $(if [ -n "$print_line_numbers" ]; then printf "%s -n"; fi) "$pattern" "$input_file" > "$output_file"
[danyaidanya-GE72NVR-78G code]$ rm test
[danyaidanya-GE72NVR-78G code]$ ./l-grep-frontend.sh -i l-grep-frontend.sh -o /dev/stdout -p getopts
while getopts "i:o:p:Cn" opt; do
[danyaidanya-GE72NVR-78G code]$ ./l-grep-frontend.sh -i l-grep-frontend.sh -o /dev/stdout -p usage
[danyaidanya-GE72NVR-78G code]$ ./l-grep-frontend.sh -i l-grep-frontend.sh -o /dev/stdout -p usage -C
echo "Usage: $0 -i <input_file> -o <output_file> -p <pattern> [-C] [-n]"
[danyaidanya-GE72NVR-78G code]$ ./l-grep-frontend.sh -i l-grep-frontend.sh -o /dev/stdout -p echo -n
21: echo "Invalid option: -$OPTARG" >&2
25: echo "Option -$OPTARG requires an argument." >&2
26: echo "Usage: $0 -i <input_file> -o <output_file> -p <pattern> [-C] [-n]"
36:grep $(if [ -n "$case_insensitive" ]; then echo "-i"; fi) $(if [ -n "$print_line_numbers" ]; then printf "%s -n"; fi) "$pattern" "$input_file" > "$output_file"
[danyaidanya-GE72NVR-78G code]$
```

Рис. 3.2: Результат работы программы поиска

Вторая программа представлена на рис. 3.3. Она состоит из двух частей – кода на языке Си и командного файла. Код на Си принимает на ввод одно число и

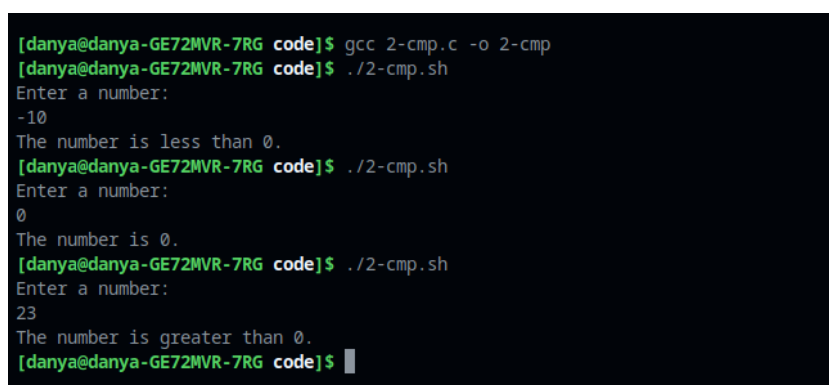
затем завершается с кодом возврата, соответствующим отношению этого числа к нулю. Командный файл пишет приглашение, запускает эту программу, затем считывает код возврата и использует его внутри case. В итоге получается вывод, как на рис. 3.4.



```
1 #include<stdio.h>
2
3 int main() {
4     int value;
5     scanf("%d", &value);
6     if (value == 0) return 0;
7     else if (value < 0) return 1;
8     else if (value > 0) return 2;
9     else return 255;
10 }
```

```
1 #!/bin/bash
2
3 echo "Enter a number:"
4 ./2-cmp
5 case $? in
6     0)
7         echo "The number is 0."
8         ;;
9     1)
10        echo "The number is less than 0."
11        ;;
12    2)
13        echo "The number is greater than 0."
14        ;;
15    *)
16        echo "Unknown error."
17        ;;
18 esac
19
```

Рис. 3.3: Программа сравнения чисел



```
[danya@danya-GE72MVR-7RG code]$ gcc 2-cmp.c -o 2-cmp
[danya@danya-GE72MVR-7RG code]$ ./2-cmp.sh
Enter a number:
-10
The number is less than 0.
[danya@danya-GE72MVR-7RG code]$ ./2-cmp.sh
Enter a number:
0
The number is 0.
[danya@danya-GE72MVR-7RG code]$ ./2-cmp.sh
Enter a number:
23
The number is greater than 0.
[danya@danya-GE72MVR-7RG code]$
```

Рис. 3.4: Результат работы программы сравнения чисел

Третья программа представлена на рис. 3.5. Сначала она удаляет все файлы с названием \*.tmp. После этого она создает несколько таких файлов, где каждый файл имеет имя <номер файла>.tmp. Проверка того, как это работает, представлена на рис. 3.6.



```

1  #!/bin/bash
2
3  if [ -z "$1" ]; then
4      echo "Usage: $0 <number>"
5      exit 1
6  fi
7
8  rm -f *.tmp
9
10 for name in $(seq 1 $1); do
11     touch "$name.tmp"
12 done
13

```

Рис. 3.5: Программа создания файлов

```

[danya@danya-GE72MVR-7RG code]$ ./3-mktmp.sh
Usage: ./3-mktmp.sh <number>
[danya@danya-GE72MVR-7RG code]$ ls | grep "tmp"
3-mktmp.sh
[danya@danya-GE72MVR-7RG code]$ ls | grep "tmp" | wc -l
1
[danya@danya-GE72MVR-7RG code]$ ./3-mktmp.sh 100
[danya@danya-GE72MVR-7RG code]$ ls | grep "tmp" | wc -l
101
[danya@danya-GE72MVR-7RG code]$ ./3-mktmp.sh 130
[danya@danya-GE72MVR-7RG code]$ ls | grep "tmp" | wc -l
131
[danya@danya-GE72MVR-7RG code]$ ./3-mktmp.sh 5
[danya@danya-GE72MVR-7RG code]$ ls | grep "tmp" | wc -l
6
[danya@danya-GE72MVR-7RG code]$ ls | grep "tmp"
1. tmp
2. tmp
3-mktmp.sh
3. tmp
4. tmp
5. tmp
[danya@danya-GE72MVR-7RG code]$

```

Рис. 3.6: Результат работы программы создания файлов

Четвертая программа представлена на рис. 3.7. Сначала она удаляет файл `archived.tar`, чтобы начать работу с новым файлом. После этого она ищет файлы в указанном каталоге, у которых дата изменения меньше семи дней назад. Каждый такой файл добавляется в архив `archived.tar`, или, если этот архив еще не создан, то он создается. Проверка того, как это работает, представлена на рис. 3.8.

```
1  #!/bin/bash
2
3  rm -f archived.tar
4
5  for file in $(find $1 -type f -mtime -7); do
6      echo $file
7      if [ -f archived.tar ]; then
8          tar rf archived.tar "$file" >> /dev/null
9      else
10         tar cf archived.tar "$file" >> /dev/null
11     fi
12 done
```

Рис. 3.7: Программа архивирования

```
[danya@danya-GE72MVR-7RG code]$ ./4-archive-recent.sh ../report/image/
../report/image/Screenshot_5.png
../report/image/Screenshot_3.png
../report/image/Screenshot_2.png
../report/image/Screenshot_4.png
../report/image/Screenshot_1.png
../report/image/Screenshot_6.png
[danya@danya-GE72MVR-7RG code]$ tar tf ./archived.tar
report/image/Screenshot_5.png
report/image/Screenshot_3.png
report/image/Screenshot_2.png
report/image/Screenshot_4.png
report/image/Screenshot_1.png
report/image/Screenshot_6.png
[danya@danya-GE72MVR-7RG code]$ touch -d "2020/01/01" ../report/image/*
[danya@danya-GE72MVR-7RG code]$ touch ../report/image/Screenshot_6.png
[danya@danya-GE72MVR-7RG code]$ ./4-archive-recent.sh ../report/image/
../report/image/Screenshot_6.png
[danya@danya-GE72MVR-7RG code]$ tar tf ./archived.tar
report/image/Screenshot_6.png
[danya@danya-GE72MVR-7RG code]$
```

Рис. 3.8: Результат работы программы архивирования

## 4 Контрольные вопросы

### 1. Каково предназначение команды `getopts`?

Эта команда изменяет состояние переменных окружения, чтобы обрабатывать аргументы с командной строки. Каждый запуск этой программы поглощает один аргумент из списка и добавляет его значение в выбранную переменную окружения. Оттуда различная пользовательская логика может извлекать эти значения и обрабатывать их дальше.

### 2. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы, если не изолированы через литеральные строки или обратные слешы, обрабатываются оболочкой как список строк, равных именам файлов, которым соответствует последовательность с метасимволами. Например, если задана команда `ls test/*`, и в каталоге `test` есть файлы `foo.txt`, `bar.txt`, `baz.txt` и директория `quux`, то оболочка раскроет эту строку как `ls test/foo.txt test/bar.txt test/baz.txt test/quux`.

### 3. Какие операторы управления действиями вы знаете?

В Bash используются следующие конструкции для управления потоком выполнения команд:

- ``if/then/else/fi``
- ``case/switch/default/esac``
- ``while/do/done``

- ``until/do/done``
- ``for/in/done``

4. Какие операторы используются для прерывания цикла?

Следующие операторы могут прервать текущий цикл:

- ``break`` продолжит выполнение после конца блока цикла
- ``continue`` продолжит выполнение со следующей итерации цикла
- ``return`` прервет выполнение всей функции или программы

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

Эти команды всегда возвращают одно и тоже выходное значение и больше ничего не делают, что делает их удобными для подстановки конкретных значений в операторы. Например, бесконечный цикл начинается с `while true`. Помимо этого эти программы можно использовать в ситуациях, где нужно предоставить путь к программе, но эта программа должна не делать ничего. Например, чтобы выключить доступ к оболочке для какого-то пользователя, можно указать его оболочку как `/bin/false`.

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

Эта строка начинает блок кода, который будет выполнен только если существует файл с названием `man$s/$i.$s`, где `$s` и `$i` – переменные.

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

Эти два оператора создают циклы, но цикл `while` выполняется, пока выражение истинно, а `until` выполняется, пока выражение ложно. Таким образом, `while <условие>` – это аналог `until !<условие>`.

## 5 Выводы

В этой лабораторной работе мы познакомились с основами программирования в Bash. Поскольку командный интерпретатор есть всегда, полезно уметь использовать его возможности не только в интерактивном режиме, но и для написания программ. Те четыре программы, которые мы написали, каждая демонстрируют один из важных компонентов работы с командным интерпретатором.