

Операционные системы

Лабораторная работа №12

Гульдяев Тихон Дмитриевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	12
4	Ответы на контрольные вопросы	13
	Список литературы	15

Список таблиц

Список иллюстраций

2.1	Код первой программы	7
2.2	Пример использования первой программы	8
2.3	Код второй программы и скрипта	8
2.4	Пример использования второй программы и скрипта	9
2.5	Код третьей программы	9
2.6	Пример использования третьей программы	10
2.7	Код четвертой программы	10
2.8	Пример использования четвертой программы	11
2.9	Второй пример использования четвертой программы	11

1. Цель работы

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

2. Выполнение лабораторной работы

Первая программа:

Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами: `-i inputfile` — прочитать данные из указанного файла; `-o outputfile` — вывести данные в указанный файл; `-r 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 `-r`.

Код первой программы. (рис. 2.1).

```

$ grep.sh
1  #!/bin/bash
2
3  input_file=""
4  output_file=""
5  pattern=""
6  sens=0
7  numbers=0
8  while getopts "i:o:p:Cn" opt; do
9      case $opt in
10         i) input_file=$OPTARG;;
11         o) output_file=$OPTARG;;
12         p) pattern=$OPTARG;;
13         C) sens=1;;
14         n) numbers=1;;
15         \?) echo "Неверный ключ: -$OPTARG" >&2 exit 1;;
16     esac
17 done
18 if [ -z "$pattern" ]; then
19     echo "Не указан паттерн"
20     exit 1
21 fi
22 if [ -z $input_file ]; then
23     echo "Не указан файл"
24     exit 1
25 fi
26 grep_command="grep "
27 if [ $sens -eq 1 ]; then
28     grep_command+=" -i"
29 fi
30 if [ $numbers -eq 1 ]; then
31     grep_command+=" -n"
32 fi
33 grep_command+=" $pattern"
34 if [ -n "$output_file" ]; then
35     $grep_command "$input_file" > "$output_file"
36 else
37     $grep_command "$input_file"
38 fi

```

Рис. 2.1: Код первой программы

Пример использования первой программы. (рис. 2.2).

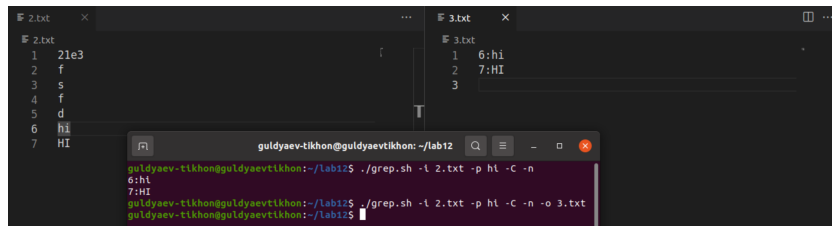


Рис. 2.2: Пример использования первой программы

Вторая программа и скрипт:

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.

Код второй программы и скрипта (рис. 2.3)

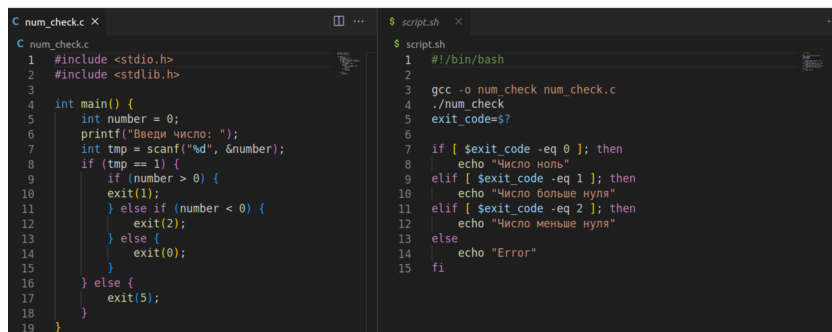



Рис. 2.3: Код второй программы и скрипта

Пример использования второй программы и скрипта. (рис. 2.4).




```
guldyaev-tikhon@guldyaevtikhon: ~/lab12
guldyaev-tikhon@guldyaevtikhon:~/lab12$ ./script.sh
Введи число: 1
Число больше нуля
guldyaev-tikhon@guldyaevtikhon:~/lab12$ ./script.sh
Введи число: -1
Число меньше нуля
guldyaev-tikhon@guldyaevtikhon:~/lab12$ ./script.sh
Введи число: 0
Число ноль
guldyaev-tikhon@guldyaevtikhon:~/lab12$ ./script.sh
Введи число: f
Error
guldyaev-tikhon@guldyaevtikhon:~/lab12$
```

Рис. 2.4: Пример использования второй программы и скрипта

Третья программа:

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).

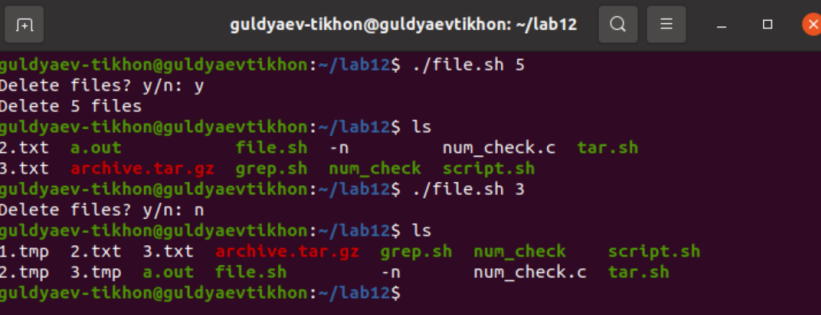
Код третьей программы. (рис. 2.5).



```
$ file.sh
1  #!/bin/bash
2
3  if [ $# -ne 1 ] || [ "$1" -le 0 ]; then
4      exit 1
5  fi
6
7  for ((i=1;i<=$1;i++)); do
8      touch "$i.tmp"
9  done
10
11 read -p "Delete files? y/n: " del
12 if [ $del == 'y' ]; then
13     for ((i=1;i<=$1;i++)); do
14         rm "$i.tmp"
15     done
16     echo "Delete $1 files"
17 fi
```

Рис. 2.5: Код третьей программы

Пример использования третьей программы. (рис. 2.6).



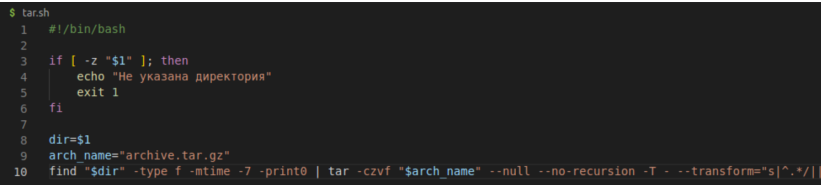
```
guldyaev-tikhon@guldyaevtkhon: ~/lab12
guldyaev-tikhon@guldyaevtkhon:~/lab12$ ./file.sh 5
Delete files? y/n: y
Delete 5 files
guldyaev-tikhon@guldyaevtkhon:~/lab12$ ls
2.txt  a.out      file.sh  -n      num_check.c  tar.sh
3.txt  archive.tar.gz  grep.sh  num_check  script.sh
guldyaev-tikhon@guldyaevtkhon:~/lab12$ ./file.sh 3
Delete files? y/n: n
guldyaev-tikhon@guldyaevtkhon:~/lab12$ ls
1.tmp  2.txt  3.txt  archive.tar.gz  grep.sh  num_check  script.sh
2.tmp  3.tmp  a.out  file.sh        -n      num_check.c  tar.sh
guldyaev-tikhon@guldyaevtkhon:~/lab12$
```

Рис. 2.6: Пример использования третьей программы

Четвертая программа:

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)

Код четвертой программы. (рис. 2.7).



```
$ tar.sh
1 #!/bin/bash
2
3 if [ -z "$1" ]; then
4     echo "Не указана директория"
5     exit 1
6 fi
7
8 dir=$1
9 arch_name="archive.tar.gz"
10 find "$dir" -type f -mtime -7 -print0 | tar -czvf "$arch_name" --null --no-recursion -T - --transform="s|^.*/||"
```

Рис. 2.7: Код четвертой программы

Пример использования четвертой программы. (рис. 2.8).

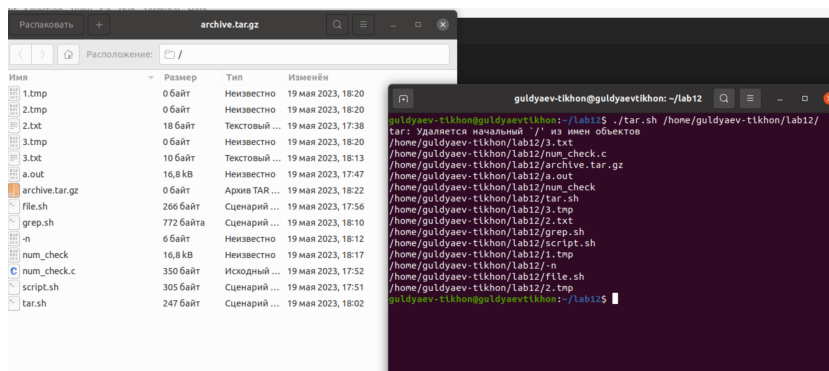


Рис. 2.8: Пример использования четвертой программы

Второй пример использования четвертой программы. (рис. 2.9).

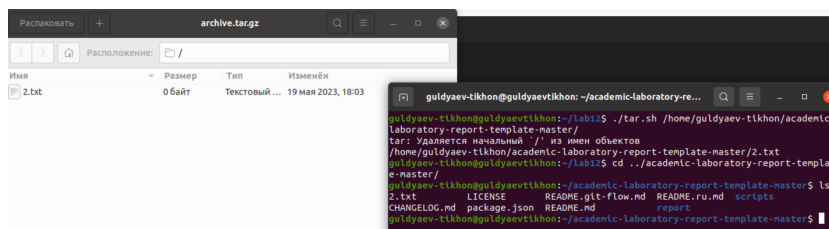


Рис. 2.9: Второй пример использования четвертой программы

3. Выводы

Я изучил основы программирования в оболочке ОС UNIX. Научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

4. Ответы на контрольные вопросы

1. Каково предназначение команды `getopts`?

Команда `getopts` используется в скриптах на языке `shell` для обработки опций командной строки. Она позволяет определить опции и их аргументы, переданные скрипту, и обработать их соответствующим образом.

2. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы (такие как `*`, `?`, `[]` и другие) используются в `шелле` для генерации имён файлов. Они позволяют сопоставлять имена файлов с определенными шаблонами и выполнять операции с соответствующими файлами.

3. Какие операторы управления действиями вы знаете?

Операторы управления действиями включают условные операторы (`if`, `case`), операторы цикла (`for`, `while`, `until`) и операторы переадресации ввода-вывода (`>`, `<`, `|` и другие). Они позволяют контролировать выполнение команд в скрипте и управлять потоками данных.

4. Какие операторы используются для прерывания цикла?

Для прерывания цикла в языке `shell` используются операторы `break` и `continue`. Оператор `break` прерывает выполнение цикла и передает управление за пределы цикла, а оператор `continue` прерывает текущую итерацию цикла и переходит к следующей итерации.

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

Команда `false` возвращает значение “ложь” (код возврата 1), а команда `true` возвращает значение “истина” (код возврата 0). Обычно они используются для создания заглушек

или фиктивных команд, которые всегда возвращают ожидаемые значения, независимо от выполняемых действий.

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

Данная строка является условием проверки в командном файле (скрипте). В ней используется команда `test` с опцией `-f`, которая проверяет, является ли файл с именем, сформированным из переменных `$s`, `$i` и `$s`, обычным файлом. Если условие истинно, то выполняется соответствующий блок кода.

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

Конструкция `while` выполняет блок кода, пока условие истинно, тогда как конструкция `until` выполняет блок кода, пока условие ложно. То есть, `while` выполняет цикл, пока условие остается истинным, а `until` выполняет цикл, пока условие не станет истинным.

Список литературы

<https://www.google.ru>

<https://chat.openai.com/chat>