

# **Отчёт по лабораторной работе №11**

**Дисциплина: Операционные системы**

Подъярова Ксения Витальевна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	13
4	Ответы на контрольные вопросы	14

## Список иллюстраций

2.1	Создание файла . . . . .	6
2.2	Скрипт . . . . .	7
2.3	Скрипт . . . . .	7
2.4	Проверка работы . . . . .	8
2.5	Создание файла . . . . .	8
2.6	Скрипт . . . . .	9
2.7	Скрипт . . . . .	9
2.8	Проверка работы . . . . .	10
2.9	Создание файла . . . . .	10
2.10	Скрипт . . . . .	11
2.11	Проверка работы . . . . .	11
2.12	Создание файла . . . . .	12
2.13	Скрипт . . . . .	12
2.14	Проверка работы . . . . .	12

## **Список таблиц**

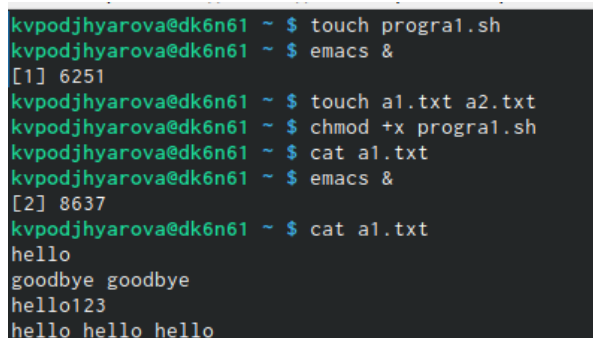
# 1 Цель работы

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

## 2 Выполнение лабораторной работы

1. 1) Используя команды `grep`, написала командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — 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`. Для данной задачи я создала файл `progra1.sh` (рис. 2.1) и написала соответствующий скрипт (рис. 2.2) , (рис. 2.3)



```
kvpodjhyarova@dk6n61 ~ $ touch progra1.sh
kvpodjhyarova@dk6n61 ~ $ emacs &
[1] 6251
kvpodjhyarova@dk6n61 ~ $ touch a1.txt a2.txt
kvpodjhyarova@dk6n61 ~ $ chmod +x progra1.sh
kvpodjhyarova@dk6n61 ~ $ cat a1.txt
kvpodjhyarova@dk6n61 ~ $ emacs &
[2] 8637
kvpodjhyarova@dk6n61 ~ $ cat a1.txt
hello
goodbye goodbye
hello123
hello hello hello
```

Рис. 2.1: Создание файла

```
#!/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0;
while getopts i:o:p:Cn optletter
do case $optletter in
    i) iflag=1; ival=$OPTARG;;
    o) oflag=1; oval=$OPTARG;;
    p) pflag=1; pval=$OPTARG;;
    C) Cflag=1;;
    n) nflag=1;;
    *) echo illegal option $optletter
    esac
done
if ((pflag==0))
then
    echo "Шаблон не найден"
    exit
fi
if ((iflag==0))
then
    echo "Входящий файл не найден"
    exit
fi
if ((oflag==0))
then
    echo "Исходящий файл не найден"
    exit
fi
```

Рис. 2.2: Скрипт

```
if ((nflag==0))
then
    if ((iflag==0))
    then
        grep $pval $ival > $oval
    else
        grep -i $pval $ival > $oval
    fi
else
    if ((iflag==0))
    then
        grep -n $pval $ival > $oval
    else
        grep -i -n $pval $ival > $oval
    fi
fi
```

Рис. 2.3: Скрипт

- 2) Проверила работу написанного скрипта, используя различные опции (например команду `./progra1.sh -i a1.txt -o a2.txt -C -n`), предварительно добавив право на исполнение файла (`chmod +x progra1.sh`) и создав 2 файла, которые необходимы для выполнения программы (`a1.txt`, `a2.txt`). Скрипт работает корректно. (рис. 2.4)

```

kvpodjhyarova@dk6n61 ~ $ ./progra1.sh -i a1.txt -o a2.txt -p hello -n
kvpodjhyarova@dk6n61 ~ $ cat a2.txt
1:hello
3:hello123
4:hello hello hello
kvpodjhyarova@dk6n61 ~ $ ./progra1.sh -i a1.txt -o a2.txt -p hello -C -n
kvpodjhyarova@dk6n61 ~ $ cat a2.txt
1:hello
3:hello123
4:hello hello hello
kvpodjhyarova@dk6n61 ~ $ ./progra1.sh -o a2.txt -p hello -C -n
Входящий файл не найден

```

Рис. 2.4: Проверка работы

2. 1) Написала 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. Для данной задачи я создала 2 файла: `chslo.c` `chislo.sh` (рис. 2.5) и написала соответствующие скрипты (рис. 2.6) (рис. 2.7)

```

kvpodjhyarova@dk6n61 ~ $ touch chslo.c
kvpodjhyarova@dk6n61 ~ $ touch chislo.sh
kvpodjhyarova@dk6n61 ~ $ emacs &
[2] 16543

```

Рис. 2.5: Создание файла



```
#!/bin/bash
gcc chslo.c -o chslo
./chslo
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0"
esac
```

J:\*\*\*- **chislo.sh** All L10 (

Рис. 2.6: Скрипт

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("Введите число\n");
    int a;
    scanf("%d", &a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a==0) exit(2);
    return 0;
}
```

J:--- **chslo.c** All L11

Рис. 2.7: Скрипт

- 2) Проверила работу написанных скриптов (команда ./chislo.sh), предварительно добавив право на исполнение файла (chmod +x chislo.sh). Скрипты

работают корректно.(рис. 2.8)

```
kvpodjhyarova@dk6n61 ~ $ chmod +x chislo.sh
kvpodjhyarova@dk6n61 ~ $ ./chislo.sh
Введите число
0
Число равно 0
kvpodjhyarova@dk6n61 ~ $ ./chislo.sh
Введите число
9
Число больше 0
kvpodjhyarova@dk6n61 ~ $ ./chislo.sh
Введите число
-25
Число меньше 0
kvpodjhyarova@dk6n61 ~ $
```

Рис. 2.8: Проверка работы

3. 1) Написала 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, передается в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). Для данной задачи я создала файл files.sh (рис. 2.9) и написала соответствующий скрипт (рис. 2.10)

```
kvpodjhyarova@dk6n61 ~ $ touch files.sh
kvpodjhyarova@dk6n61 ~ $ emacs &
[3] 17900
```

Рис. 2.9: Создание файла



```

kvpodjhyarova@dk6n61 ~ $ touch pr4.sh
kvpodjhyarova@dk6n61 ~ $ emacs &
[4] 19720

```

Рис. 2.12: Создание файла

```

#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files"; do
    files=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing

```

Рис. 2.13: Скрипт

- 2) Далее я проверила работу написанного скрипта, предварительно добавив право на исполнение файла и создав отдельный каталог с несколькими файлами ((рис. 2.11)

```

kvpodjhyarova@dk6n61 ~ $ chmod +x pr4.sh
[3]- Завершён emacs
kvpodjhyarova@dk6n61 ~ $ mkdir Catalog1
kvpodjhyarova@dk6n61 ~ $ cd Catalog1
kvpodjhyarova@dk6n61 ~/Catalog1 $ ~/pr4.sh
./
tar: ./Catalog1.tar: файл является архивом; не сброшен
kvpodjhyarova@dk6n61 ~/Catalog1 $ tar -tf Catalog1.tar
./
kvpodjhyarova@dk6n61 ~/Catalog1 $ ./pr4.sh
bash: ./pr4.sh: Нет такого файла или каталога
kvpodjhyarova@dk6n61 ~/Catalog1 $ ~/pr4.sh
./
tar: ./Catalog1.tar: файл является архивом; не сброшен
tar: ./Catalog1.tar: файл является архивом; не сброшен
kvpodjhyarova@dk6n61 ~/Catalog1 $ tar -tf Catalog1.tar
./
kvpodjhyarova@dk6n61 ~/Catalog1 $

```

Рис. 2.14: Проверка работы

## **3 Выводы**

В ходе выполнения лабораторной работы я изучила основы программирования в оболочке ОС UNIX и научилась писать небольшие командные файлы.

## 4 Ответы на контрольные вопросы

1. Команда `getopts` 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`. 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 введенные данные с помощью оператора `case`. Функция `getopts` включает две специальные переменные среды `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.
2. При перечислении имен файлов текущего каталога можно использовать следующие символы:
3. `-` соответствует произвольной, в том числе и пустой строке;
4. `?` – соответствует любому одинарному символу;

5. [c1-c2] – 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.
  - echoproг.?–выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются прог..
  - [a-z]–соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.
3. 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.
4. Два несложных способа позволяют вам прерывать циклы в оболочке bash. Команда break завершает выполнение цикла, а команда continue завершает

данную итерацию блока операторов. Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестает быть правильным. Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

5. Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования `bash`: это команда `true`, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда `false`, которая всегда возвращает код завершения, неравный нулю (т.е. ложь). Примеры бесконечных циклов: `while true do echo hello andy done` `until false do echo hello mike done`.
6. Строка `if test -f mani.s/s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задает список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задает список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлен тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задает список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь). При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны.