

# **Операционные системы**

**Лабораторная работа №12**

Безрук Мария Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Задание</b>	<b>4</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>12</b>
<b>5</b>	<b>Выводы</b>	<b>16</b>

# 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. Ознакомиться с теоретическим материалом.
2. Изучить основы программирования в оболочке ОС UNIX/Linux.
3. Выполнить упражнения.
4. Ответить на контрольные вопросы.

## 3 Выполнение лабораторной работы

1)Используя команды `getopts``grep`, написалакомандный 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.

Для данной задачи я создала файл `prog1.sh` и написала соответствующие скрипты.

```
#!/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"
    if (($iflag==0))
    then echo " Файл не найден"
    else
        if (($oflag==0))
        then if (($Cflag==0))
            then if (($nflag==0))
                then grep $pval $ival
                else grep -n $pval $ival
            fi
        fi
    fi
fi
```

Figure 3.1: Первый скрипт

```

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

```

Figure 3.2: Продолжение скрипта

Далее я проверила работу написанного скрипта, используя различные опции (например, команда «./prog.sh -l a1.txt -o a2.txt -p capital -C -n»), предварительно добавив право на исполнение файла (команда «chmod +x prog1.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt

```

mmbezruk@dk4n64 ~$ touch prog1.sh
mmbezruk@dk4n64 ~$ emacs &
[1] 3128
mmbezruk@dk4n64 ~$ touch a1.txt a2.txt
mmbezruk@dk4n64 ~$ emacs &
[2] 6546
mmbezruk@dk4n64 ~$ cat a1.txt
Moscow is the capital of Russia
asdfgh asdfg
aqws
asdf ghjkl

```

Figure 3.3: Проверка скрипта

```

mmbezruk@dk4n64 ~$ chmod +x prog1.sh
mmbezruk@dk4n64 ~$ cat a1.txt
Moscow is the capital of Russia
asdfgh asdfg
aqws
asdf ghjkl
mmbezruk@dk4n64 ~$ ./prog1.sh a1.txt -o a2.txt -p capital -C -n
Шаблон не найден
Файл не найден
mmbezruk@dk4n64 ~$ cat a2.txt
London is the capital of UK
ghjk hjk hjk
fghj
fghjk hjkkl
mmbezruk@dk4n64 ~$ ./prog1.sh -i a1.txt -o a2.txt -p hj -n
mmbezruk@dk4n64 ~$ cat a2.txt
London is the capital of UK
ghjk hjk hjk
fghj
fghjk hjkkl
mmbezruk@dk4n64 ~$ ./prog1.sh -i a1.txt -C -n
Шаблон не найден
./prog1.sh -o a2.txt -p capital -C -n
touch chislo.c

```

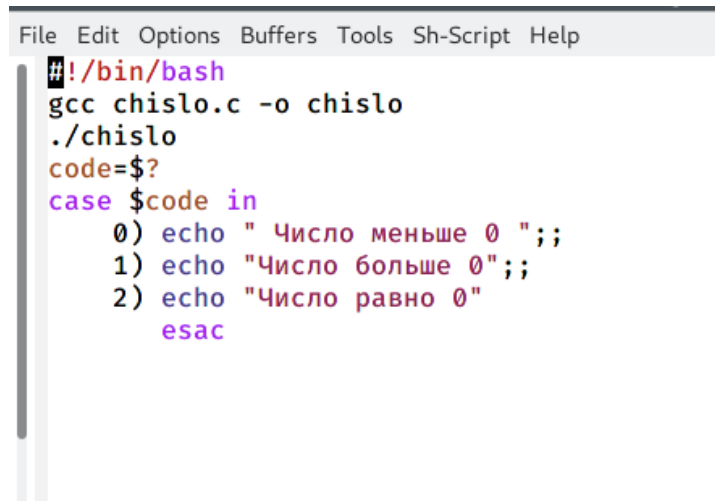
Figure 3.4: Второй скрипт

Скрипт работает корректно.

- 2) 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: `chislo.c` и `chislo.sh` и написала соответствующие скрипты.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     printf("Введите число \n");
6     int a;
7     scanf("%d" , &a);
8     if (a<0) exit(0);
9     if (a>0) exit(1);
0     if (a==0) exit(2);
1     return 0;
2 }
```

Figure 3.5: Третий скрипт



```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
gcc chislo.c -o chislo
./chislo
code=$?
case $code in
  0) echo " Число меньше 0 ";;
  1) echo "Число больше 0";;
  2) echo "Число равно 0"
esac
```

Figure 3.6: Третий скрипт

Далее я проверила работу написанных скриптов (команда «./chislo.sh»), предварительно добавив право на исполнение файла (команда «chmod+x chislo.sh»)



```
Файл  Правка  Вид  Закладки  Настройка  Справка
mmbezruk@dk4n64 - $ touch chislo.c
mmbezruk@dk4n64 - $ touch chislo.sh
mmbezruk@dk4n64 - $ emacs &
[1] 10870
mmbezruk@dk4n64 - $ chmod +x chislo.sh
[1]+  Завершён      emacs
mmbezruk@dk4n64 - $ ./chislo.sh
Введите число
0
Число равно 0
mmbezruk@dk4n64 - $ ./chislo.sh
Введите число
4
Число больше 0
mmbezruk@dk4n64 - $ ./chislo.sh
Введите число
-9
Число меньше 0
mmbezruk@dk4n64 - $ touch files.sh
mmbezruk@dk4n64 - $ emacs &
[1] 11715
mmbezruk@dk4n64 - $
```

Figure 3.7: Проверка скрипта

Скрипты работают корректно.

- 3) Написала 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). Для данной задачи я создала файл: files.sh и написала соответствующий скрипт.



Figure 3.10: Пятый скрипт

Далее я проверила работу написанного скрипта (команды «`sudo ~/prog4.sh`» и «`tar -tf Catalog1.tar`»), предварительно добавив право на исполнение файла (команда «`chmod +x prog4.sh`») и создав отдельный `catalog1` с несколькими файлами. Файлы, измененные более недели назад, заархивированы не были.

```
mmbezruk@dk4n64 ~/Catalog1 $ ls -l
итого 108548
-rw-r--r-- 1 mmbezruk studsci 111149056 мая 29 12:58 '2021-05-29 11-29-04.mkv'
-rwxr-xr-x 1 mmbezruk studsci 203 мая 29 12:31 chislo.sh
-rwxr-xr-x 1 mmbezruk studsci 256 мая 29 12:43 files.sh
-rwxr-xr-x 1 mmbezruk studsci 224 мая 29 12:51 prog4.sh
-rwxr-xr-x 1 mmbezruk studsci 708 мая 29 12:09 prog1.sh
mmbezruk@dk4n64 ~/Catalog1 $ tar -tf Catalog1.tar
```

Figure 3.11: Проверка скрипта

Скрипт работает корректно.

## 4 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 введённые 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 её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.

- 2) При перечислении имён файлов текущего каталога можно использовать следующие символы:

- `-` – соответствует произвольной, в том числе и пустой строке;
- `?` – соответствует любому одинарному символу;
- `[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`.
- `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.

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 перестаёт 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, не равный 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 mans/i.s, mans/i.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 задаёт 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 задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт 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.

## **5 Выводы**

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.