Лабораторная работа №12

Отчет

Бондарь Алексей Олегович

Содержание

List of Tables

List of Figures

1 Цель работы

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

2 Задачи

1. Познакомиться с логическими управляющими конструкций и циклов.
2. В ходе работы написать 4 командных файла.
3. Выполнить отчет.

3 Выполнение лабораторной работы

1) Используя команды getopts grep, написал командный файл, который анализирует командную строку с ключами:

- -iinputfile — прочитать данные из указанного файла;
- -ooutputfile — вывести данные в указанный файл;
- -pшаблон — 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.

Для данной задачи я создал файл prog1.sh и написал соответствующие скрипты. (рис. 3.1)

```
#!/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) oflaf=1; oval=$OPTARG;;
       p) pflag=1; pval=$OPTARG;;
       C) Cflag=1;;
       n) flag=1;;
       *) echo illegal option $optletter
   esac
done
if (($pflag==0))
then echo "Шаблон не найден"
else
    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 $ival
                  fi
             else if (($nflag==0))
                  then grep -i $pval $ival
                  else grep -i -n $pval $ival
                  fi
             fi
        else if (($Cglag==0))
             then if (($nflag==0))
                  then grep $pval $ival > $oval
                  else grep -n $pval $ival > $oval
                  fi
             else if (($nflag==0))
                  then grep -i $pval $ival > $oval
                  else -i -n $pval $ival > $oval
                  fi
             fi
        fi
    fi
fi
```

Figure 3.1: Первый скрипт

Далее я проверил работу написанного скрипта, используя различные опции (например, команда «./prog.sh –i 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. Скрипт работает корректно. (рис. 3.2):

```
aabondarj@dk6n64 ~ $ touch prog1.sh
aabondarj@dk6n64 ~ $ emacs &
[1] 3698
aabondarj@dk6n64 ~ $ touch a1.txt a2.txt
aabondarj@dk6n64 ~ $ chmod +x prog1.sh
aabondarj@dk6n64 ~ $ cat ai.txt
cat: ai.txt: Нет такого файла или каталога
aabondarj@dk6n64 ~ $ cat a1.txt
aabondarj@dk6n64 ~ $ cat a1.txt
Moscow is the capital of Russia
Paris is the Capital of France
Simple text
Rome is not the CAPITAL of Canada
aabondarj@dk6n64 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p capital -C -n
Moscow is the capital of Russia
Paris is the Capital of France
Rome is not the CAPITAL of Canada
aabondarj@dk6n64 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p capital -n
Moscow is the capital of Russia
aabondarj@dk6n64 ~ $ cat a2.txt
aabondarj@dk6n64 ~ $ cat a2.txt
aabondarj@dk6n64 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p capital -n
Moscow is the capital of Russia
aabondarj@dk6n64 ~ $ cat a2.txt
aabondarj@dk6n64 ~ $ cat a2.txt
Moscow is the capital of Russia
aabondarj@dk6n64 ~ $ ./prog1.sh -i a1.txt -C -n
Шаблон не найден
aabondarj@dk6n64 ~ $ ./prog1.sh -o a2.txt -p capital -C -n
Файл не найден
aabondarj@dk6n64 ~ $ touch chislo.c
aabondarj@dk6n64 ~ $ touch chislo.sh
```

Figure 3.2: Проверка работы скрипта

2) Написал 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 создал 2 файла: chislo.c и chislo.sh и написала соответствующие скрипты. (рис. 3.3) (рис. 3.4)

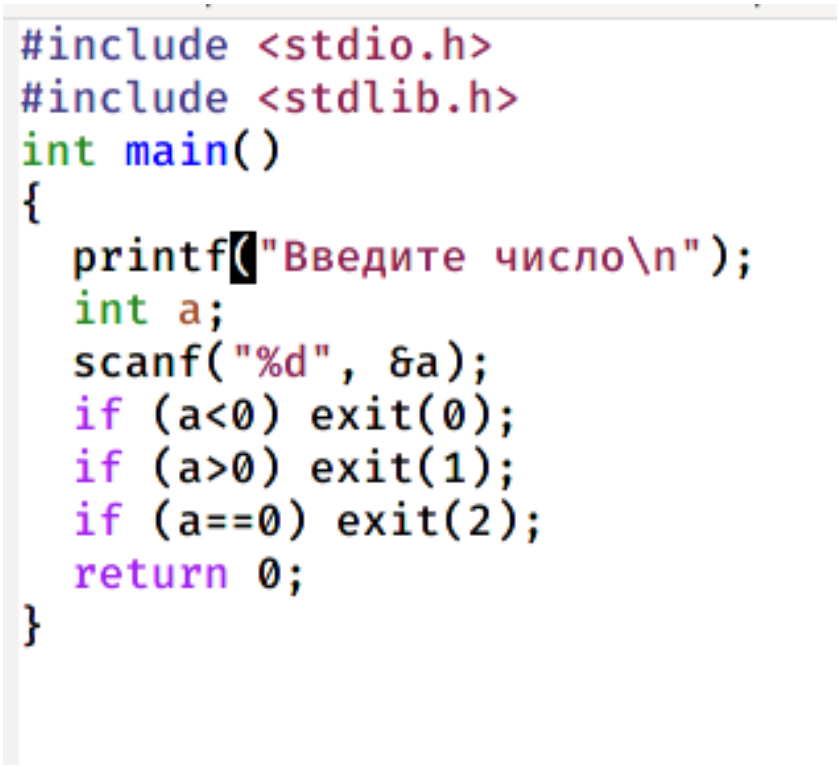


Figure 3.3: Файл

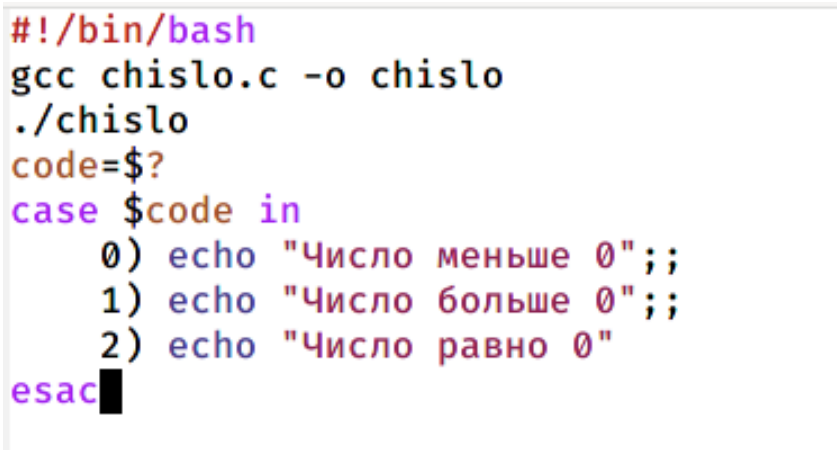


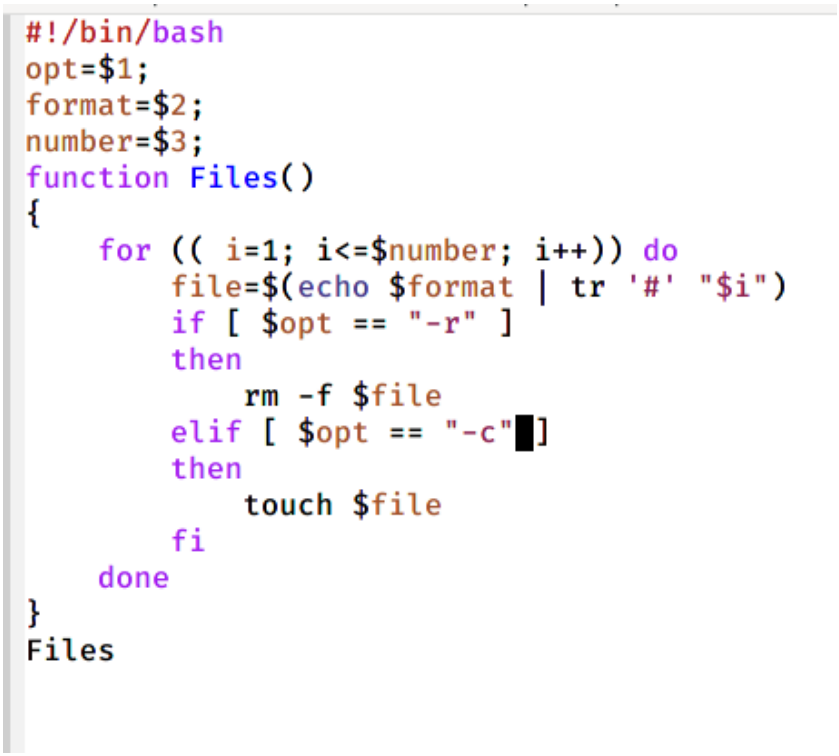
Figure 3.4: Файл

Далее я проверил работу написанных скриптов (команда «./chislo.sh»), предварительно добавив право на исполнение файла (команда «chmod +x chislo.sh»). Скрипты работают корректно. (рис. 3.5)

```
aabondarj@dk6n64 ~ $ ./chislo.sh
Введите число
0
Число равно 0
aabondarj@dk6n64 ~ $ ./chislo.sh
Введите число
5
Число больше 0
aabondarj@dk6n64 ~ $ ./chislo.sh
Введите число
-5
Число меньше 0
```

Figure 3.5: Проверка работы скриптов

3) Написал 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 создал файл files.sh и написала соответствующий скрипт. (рис. 3.6)



```
#!/bin/bash
opt=$1;
format=$2;
number=$3;
function Files()
{
    for (( i=1; i<=$number; i++)) do
        file=$(echo $format | tr '#' "$i")
        if [ $opt == "-r" ]
        then
            rm -f $file
        elif [ $opt == "-c"]
        then
            touch $file
        fi
    done
}
Files
```

Figure 3.6: Третий скрипт

Далее я проверил работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «chmod +x files.sh»). Сначала я создал три файла (команда «./files.sh –c abc#.txt 3»), удовлетворяющие условию задачи, а потом удалил их (команда «./files.sh –r abc#.txt 3»). (рис. 3.7) , (рис. 3.8) , (рис. 3.9)

```
aabondarj@dk6n64 ~ $ chmod +x files.sh
aabondarj@dk6n64 ~ $ ls
'2021-05-14 14-28-02.mkv'   lab07
 a1.txt                     lab07.asm
 a2.txt                    '#lab10.sh#'
 ARA.cpp                    lab10.sh
 asdfg                      lab2
 asdfg.asm                  lab2.asm
 backup                     lab.asm
 backup.sh                  labor
 backup.sh~                 newdir
 chislo                     prog1.sh
 chislo.c                   prog1.sh~
 chislo.c~                  prog2.sh
 chislo.sh                  prog2.sh~
 chislo.sh~                '#progls.sh#'
```

Figure 3.7: Проверка работы скрипта

```
aabondarj@dk6n64 ~ $ ./files.sh -c abc#.txt 3
aabondarj@dk6n64 ~ $ ls
'2021-05-14 14-28-02.mkv'   lab06
 a1.txt                     lab06.asm
 a2.txt                     lab07
 abc1.txt                   lab07.asm
 abc2.txt                  '#lab10.sh#'
 abc3.txt                   lab10.sh
 ARA.cpp                    lab2
 asdfg                      lab2.asm
 asdfg.asm                  lab.asm
 backup                     labor
 backup.sh                  newdir
```

Figure 3.8: Проверка работы скрипта

```
aabondarj@dk6n64 ~ $ ./files.sh -r abc#.txt 3
aabondarj@dk6n64 ~ $ ls
'2021-05-14 14-28-02.mkv'   lab07
 a1.txt                     lab07.asm
 a2.txt                    '#lab10.sh#'
 ARA.cpp                    lab10.sh
 asdfg                      lab2
 asdfg.asm                  lab2.asm
 backup                     lab.asm
 backup.sh                  labor
 backup.sh~                 newdir
 chislo                     prog1.sh
 chislo.c                   prog1.sh~
 chislo.c~                  prog2.sh
 chislo.sh                  prog2.sh~
```

Figure 3.9: Проверка работы скрипта

4) Написал командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировал его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи я создал файл prog4.sh и написал соответствующий скрипт. (рис. 3.10)

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

Figure 3.10: Четвертый скрипт

Далее я проверил работу написанного скрипта (команды «./prog4.sh»и «tar -tf catalog.tar»), предварительно добавив право на исполнение файла (команда

«chmod +x prog4.sh») и создав отдельный catalog с несколькими файлами. Скрипт работает корректно.(рис. 3.11) , (рис. 3.12)

```
aabondarj@dk6n64 ~ $ chmod +x *.sh
aabondarj@dk6n64 ~ $ mkdir lab12
aabondarj@dk6n64 ~ $ cd lab12
aabondarj@dk6n64 ~/lab12 $ ls -l
итого 0
aabondarj@dk6n64 ~/lab12 $ ls -l
итого 13
-rw-r--r-- 1 aabondarj studsci  109 мая 28 14:28 a1.txt
-rw-r--r-- 1 aabondarj studsci   32 мая 28 14:32 a2.txt
-rwxr-xr-x 1 aabondarj studsci 8072 мая 28 14:45 chislo
-rw-r--r-- 1 aabondarj studsci  196 мая 28 14:44 chislo.c
-rwxr-xr-x 1 aabondarj studsci  193 мая 28 14:42 chislo.sh
-rwxr-xr-x 1 aabondarj studsci  208 мая 28 15:11 prog4.sh
```

Figure 3.11: Проверка работы скрипта

```
aabondarj@dk6n64 ~/lab12 $ ./prog4.sh
a1.txt
chislo
chislo.c
chislo.sh
prog4.sh
a2.txt
files.sh
aabondarj@dk6n64 ~/lab12 $ tar -tf lab12.tar
a1.txt
chislo
chislo.c
chislo.sh
prog4.sh
a2.txt
files.sh
```

Figure 3.12: Проверка работы скрипта

4 Контрольные вопросы

1) Команда getopts осуществляет синтаксический анализ командной строки, выделяя флаги, ииспользуется 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 – эт осписок 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. Еслик оманда 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 доплнительное 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) Приперечислении имён файлов текущего каталога можно использовать следующие символы:

1. *–соответствует произвольной, в том числе и пустой строке;

2. ?–соответствует любому одинарному символу;

3. [c1-c2] – соответствует любому символу, лексикографически находящемуся между символами c1 и c2.

Например,

1.1. echo* – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls;

1.2. ls*.c–выведет все файлы с последними двумя символами, совпадающими с.с.

1.3. echoprog.?–выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog..

1.4. [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 OCUNIX 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 OCUNIX 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 iftest-fman$s$/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 изучил основы программирования в оболочке ОС UNIX и научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.