

# Цель работы

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

## Задание

Написать командные файлы с использованием логических управляющих конструкций и циклов.

## Выполнение лабораторной работы

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;

-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 создал файл `prog12.sh` и написал соответствующие скрипты. (рис. -@fig:001) (рис. -@fig:002 ).

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

{ #fig:001 width=70% }

Далее я проверил работу написанного скрипта, используя различные опции (например, команда «./prog12.sh -i a1.txt -o a2.txt -р capital -C -n»), предварительно добавив право на исполнение файла

(команда «chmod +x prog12.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt. Скрипт работает корректно (рис. -@fig:003).

```

aagongcharov@aagongcharov1-VirtualBox:~$ touch prog12.sh
aagongcharov@aagongcharov1-VirtualBox:~$ enacs
aagongcharov@aagongcharov1-VirtualBox:~$ touch a1.txt a2.txt
aagongcharov@aagongcharov1-VirtualBox:~$ chmod +x *.sh
aagongcharov@aagongcharov1-VirtualBox:~$ cat a1.txt
moscow
paris
london
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -o a2.txt city -C -n
Шаблон не найден
aagongcharov@aagongcharov1-VirtualBox:~$ chmod +x prog12.sh
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -o a2.txt city -C -n
Шаблон не найден
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -o a2.txt city -C -n
Шаблон не найден
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -o a2.txt city -C -n
Шаблон не найден
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -o a2.txt -p city -C -n
aagongcharov@aagongcharov1-VirtualBox:~$ cat a2.txt
1:moscow city of Russia
2:paris city of France
3:london CITY of England
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -o a2.txt -p city -n
aagongcharov@aagongcharov1-VirtualBox:~$ cat a2.txt
1:moscow city of Russia
2:paris city of France
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -i a1.txt -C -n
Шаблон не найден
aagongcharov@aagongcharov1-VirtualBox:~$ ./prog12.sh -o a2.txt -p city -C -n
Файл не найден
aagongcharov@aagongcharov1-VirtualBox:~$

```

{ #fig:003 width=70% }

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 написал соответствующие скрипты. (рис. -@fig:004) (рис. -@fig:005).

```

#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;
}

```

{ #fig:004 width=70% }

```

#!/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

```

{ #fig:005 width=70% }

Далее я проверил работу написанных скриптов (команда «./chislo.sh»), предварительно добавив право на исполнение файла (команда «chmod +x chislo.sh»). Скрипты работают корректно (рис. -@fig:006).

```

aagoncharov1@aagoncharov1-VirtualBox:~$
aagoncharov1@aagoncharov1-VirtualBox:~$ chmod +x *.sh
aagoncharov1@aagoncharov1-VirtualBox:~$ ./chislo.sh
Введите число
2
Число больше 0
aagoncharov1@aagoncharov1-VirtualBox:~$ ./chislo.sh
Введите число
0
Число равно 0
aagoncharov1@aagoncharov1-VirtualBox:~$ ./chislo.sh
Введите число
-3
Число меньше 0
aagoncharov1@aagoncharov1-VirtualBox:~$

```

{ #fig:006 width=70% }

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: file.sh и написала соответствующий скрипт (рис. -@fig:007).

```

#!/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

```

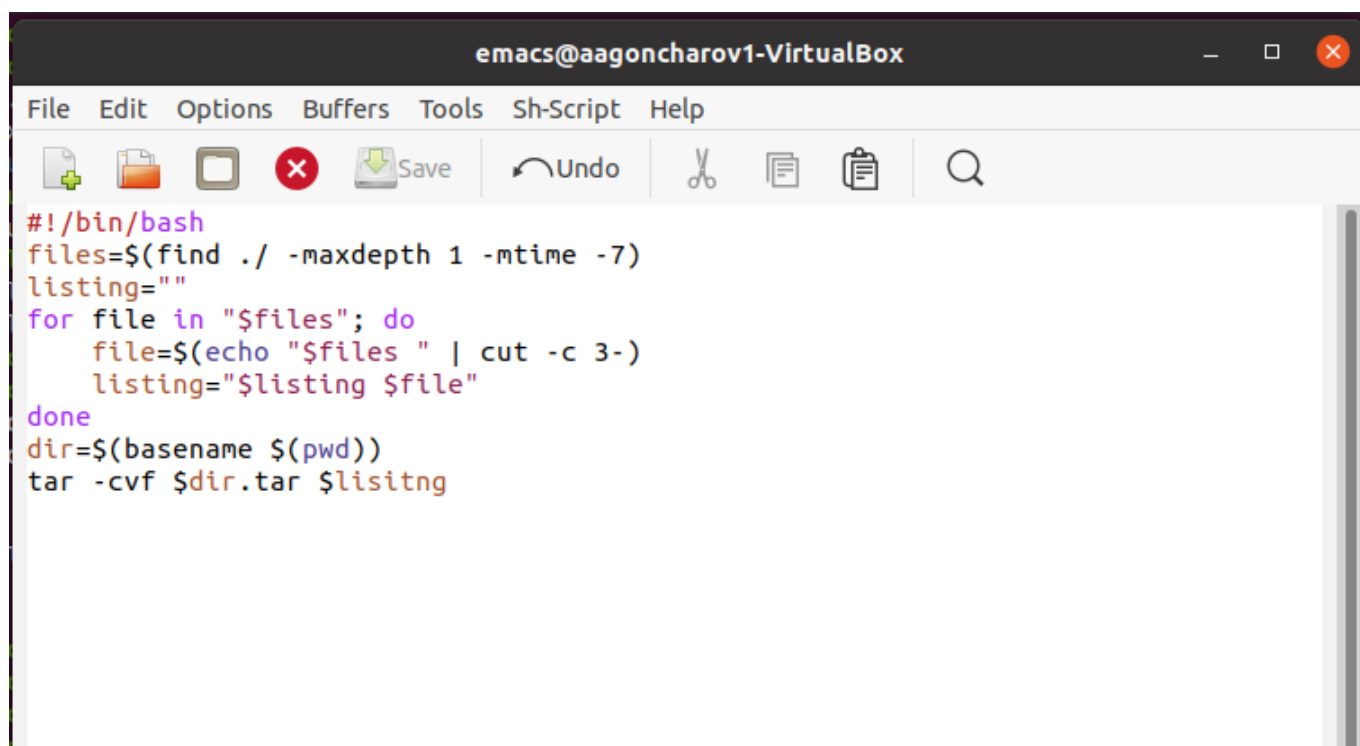
{ #fig:007 width=70% }

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 hi#.txt 3»), удовлетворяющие условию задачи, а потом удалил их (команда «./files.sh -r hi#.txt 3») (рис. -@fig:008).

```
aagoncharov1@aagoncharov1-VirtualBox:~$ emacs file.sh
aagoncharov1@aagoncharov1-VirtualBox:~$ ./file.sh -c hi#.txt 3
aagoncharov1@aagoncharov1-VirtualBox:~$ ls
al2.txt  backup.sh-  chislo.sh-  format.sh-  hi3.txt  'lab09.1.txt#'  'lab09.3.txt#'  lab13.sh-  prog2.sh-  work  Изображения  Шаблоны
al.txt  backup.sh-  chislo.sh-  format.sh-  'lab07.sh#'  'lab09.1.txt'  'lab09.3.txt'  lab13.sh-  prog2.sh-  Видео  Музыка
backup  chislo.c-  file.sh-  h11.txt  lab07.sh  'lab09.2.txt#'  'lab09.txt#'  prog12.sh-  prog1s.sh-  Документы  Общедоступные
backup.sh  chislo.c-  file.sh-  h12.txt  lab07.sh-  lab09.2.txt  lab09.txt  prog12.sh-  prog1s.sh-  Загрузки  'Рабочий стол'
```

{ #fig:008 width=70% }

Написал командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи я создал файл: tar.sh и написал соответствующий скрипт (рис. -@fig:009).



```
emacs@aagoncharov1-VirtualBox
File Edit Options Buffers Tools Sh-Script Help
+ Save Undo
#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
listing=""
for file in "$files"; do
    file=$(echo "$files" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

{ #fig:009 width=70% }

Далее я проверил работу написанного скрипта (команды «sudo ~/tar.sh»), предварительно добавив право на исполнение файла (команда «chmod +x tar.sh»). Скрипт работает корректно (рис. -@fig:010) (рис. -@fig:011).

```

aagoncharov1@aagoncharov1-VirtualBox:~$ touch tar.sh
aagoncharov1@aagoncharov1-VirtualBox:~$ enacs tar.sh
aagoncharov1@aagoncharov1-VirtualBox:~$ ls -l
итого 152
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 47 мая 28 17:19 al2.txt
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 66 мая 28 17:18 al.txt
drwxrwxr-x 2 aagoncharov1 aagoncharov1 4096 мая 28 15:40 backup
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 113 мая 28 15:26 backup.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 28 15:22 backup.sh-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 16840 мая 29 22:24 chislo
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 195 мая 29 22:23 chislo.c
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 195 мая 28 17:27 chislo.c-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 195 мая 29 22:22 chislo.sh
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 195 мая 28 17:33 chislo.sh-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 243 мая 29 23:15 rfile.sh
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 245 мая 29 23:01 file.sh-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 221 мая 28 16:04 format.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 28 16:00 format.sh-
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 11 мая 21 10:34 '#lab07.sh#'
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 98 мая 21 10:14 lab07.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 111 мая 18 20:43 lab07.sh-
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 7 мая 21 10:40 '#lab09.1.txt#'
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 21 10:31 lab09.1.txt
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 16 мая 21 10:40 '#lab09.2.txt#'
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 21 10:31 lab09.2.txt
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 13 мая 21 10:40 '#lab09.3.txt#'
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 21 10:31 lab09.3.txt
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 22 мая 21 10:40 '#lab09.txt#'
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 21 10:31 lab09.txt
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 724 мая 29 01:43 lab13.sh
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 332 мая 29 01:35 lab13.sh-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 912 мая 28 17:07 prog12.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 28 16:53 prog12.sh-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 66 мая 28 15:48 prog2.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 28 15:48 prog2.sh-
-rwxrwxr-x 1 aagoncharov1 aagoncharov1 428 мая 28 15:56 proqls.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 28 15:50 proqls.sh-
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 210 мая 29 23:23 tar.sh
-rw-rw-r-- 1 aagoncharov1 aagoncharov1 0 мая 29 23:19 tar.sh-
drwxrwxr-x 3 aagoncharov1 aagoncharov1 4096 мая 18 19:32 work
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 Видео
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 Документы
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 Загрузки
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 18 19:55 Изображения
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 Музыка
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 Общедоступные
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 'Рабочий стол'
drwxr-xr-x 2 aagoncharov1 aagoncharov1 4096 мая 4 15:31 Шаблоны
aagoncharov1@aagoncharov1-VirtualBox:~$

```

```
{ #fig:010 width=70% }
```

## 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 man$s/$i.$s` проверяет, существует ли файл `man$s/$i.$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.

## 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