

# **Лабораторная работа №11.**

**Программирование в командном процессоре ОС UNIX. Ветвления и циклы.**

Александр Андреевич Шуплецов

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение работы	7
4	Контрольные вопросы	11
5	Выводы	15
	Список литературы	16

## Список иллюстраций

3.1	текст командного файла, анализирующего командную строку с ключами . . . . .	7
3.2	проверка командного файла, анализирующего командную строку с ключами . . . . .	8
3.3	текст командного файла, выводящего число . . . . .	8
3.4	проверка командного файла, выводящего число . . . . .	9
3.5	текст командного файла, создающего указанное число файлов . .	9
3.6	проверка командного файла, создающего указанное число файлов	9
3.7	текст командного файла, который запаковывает в архив недавно использованные файлы . . . . .	10
3.8	проверка командного файла, который запаковывает в архив недавно использованные файлы . . . . .	10

## Список таблиц

# 1 Цель работы

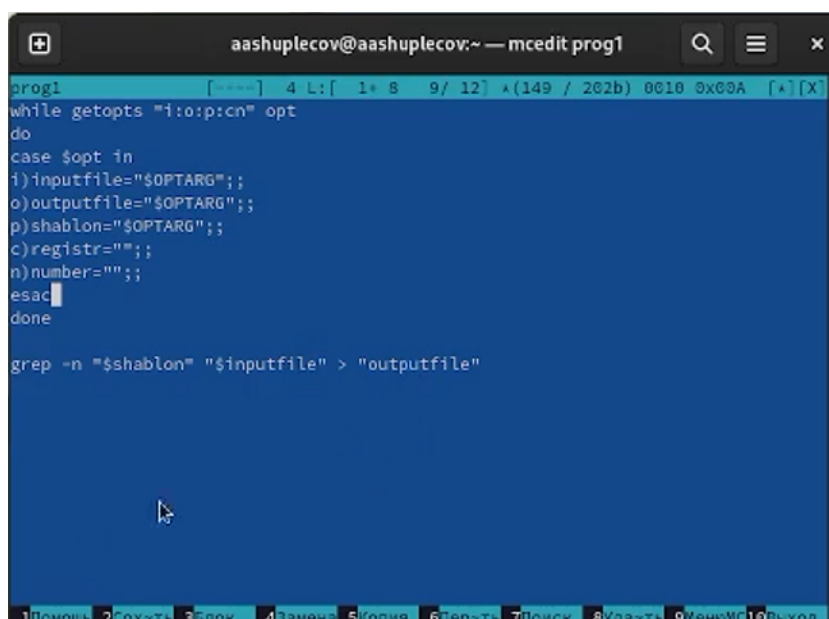
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 2 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

### 3 Выполнение работы

1. Используя команды `getopts` `grep`, напомним командный файл, который анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла; `-ooutputfile` — вывести данные в указанный файл; `-rшаблон` — 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`.

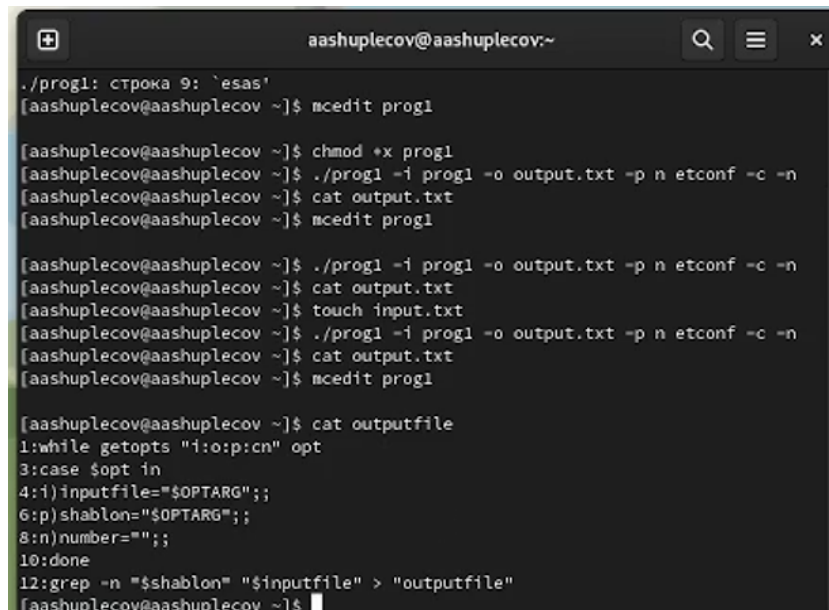


```
prog1 [----] 4 L: [ 1* 8 9/ 12] *(149 / 202b) 0010 0x00A [*][X]
while getopts "i:o:p:cn" opt
do
case $opt in
i)inputfile="$OPTARG";;
o)outputfile="$OPTARG";;
p)shablon="$OPTARG";;
c)registr="";;
n)number="";;
esac
done

grep -n "$shablon" "$inputfile" > "outputfile"
```

Рис. 3.1: текст командного файла, анализирующего командную строку с ключами

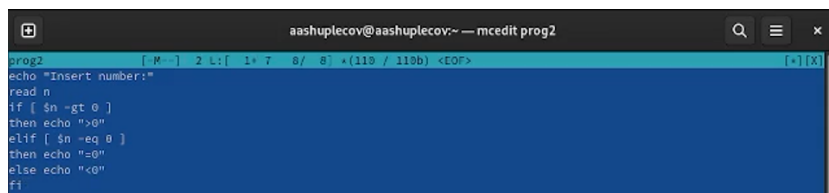
2. Убедимся, что командный файл, анализирующий командную строку с ключами, работает.



```
aashuplecov@aashuplecov:~  
./prog1: строка 9: `esas'  
[aashuplecov@aashuplecov ~]$ mcedit prog1  
  
[aashuplecov@aashuplecov ~]$ chmod +x prog1  
[aashuplecov@aashuplecov ~]$ ./prog1 -i prog1 -o output.txt -p n etconf -c -n  
[aashuplecov@aashuplecov ~]$ cat output.txt  
[aashuplecov@aashuplecov ~]$ mcedit prog1  
  
[aashuplecov@aashuplecov ~]$ ./prog1 -i prog1 -o output.txt -p n etconf -c -n  
[aashuplecov@aashuplecov ~]$ cat output.txt  
[aashuplecov@aashuplecov ~]$ touch input.txt  
[aashuplecov@aashuplecov ~]$ ./prog1 -i prog1 -o output.txt -p n etconf -c -n  
[aashuplecov@aashuplecov ~]$ cat output.txt  
[aashuplecov@aashuplecov ~]$ mcedit prog1  
  
[aashuplecov@aashuplecov ~]$ cat outputfile  
1:while getopts "i:o:p:cn" opt  
3:case $opt in  
4:i)inputfile="$OPTARG";;  
6:p)shablon="$OPTARG";;  
8:n)number="";;  
10:done  
12:grep -n "$shablon" "$inputfile" > "outputfile"  
[aashuplecov@aashuplecov ~]$
```

Рис. 3.2: проверка командного файла, анализирующего командную строку с ключами

3. Напишем на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.



```
aashuplecov@aashuplecov:~ — mcedit prog2  
prog2 [ M-- ] 2 L: [ 1+ 7 8/ 8] *(110 / 110b) <EOF> [ + ] [ X ]  
echo "Insert number:"  
read n  
if [ $n -gt 0 ]  
then echo ">0"  
elif [ $n -eq 0 ]  
then echo "=0"  
else echo "<0"  
fi
```

Рис. 3.3: текст командного файла, выводящего число

4. Убедимся, что программа, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю, работает.



```

aashuplecov@aashuplecov:~$ mcedit prog2
[aashuplecov@aashuplecov ~]$ chmod +x prog2
[aashuplecov@aashuplecov ~]$ ./prog2
Insert number:
3
>0
[aashuplecov@aashuplecov ~]$ ./prog2
Insert number:
0
=0
[aashuplecov@aashuplecov ~]$ ./prog2
Insert number:
-1
<0
[aashuplecov@aashuplecov ~]$

```

Рис. 3.4: проверка командного файла, выводящего число

5. Напишем командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $\infty$  (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.).

```

aashuplecov@aashuplecov:~$ mcedit prog3
prog3 4 L: 1+ 8 7/ 7; *(163 / 163b) <EOF>
while getopts "c:r" opt
do
case $opt in
c)n="$OPTARG"; for i in $(seq 1 $n); do touch "$i.tmp"; done;;
r)for i in $(find -name "*.tmp"); do rm $i; done;;
esac
done

```

Рис. 3.5: текст командного файла, создающего указанное число файлов

6. Убедимся, что командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $\infty$  (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.), работает.

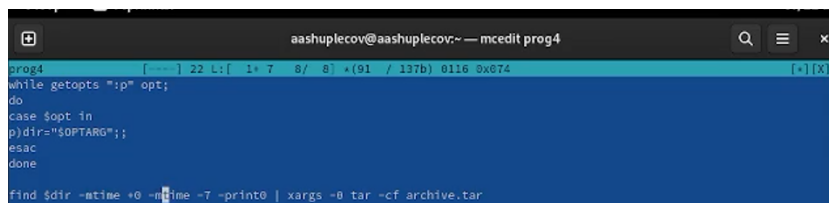
```

aashuplecov@aashuplecov:~$ mcedit prog3
[aashuplecov@aashuplecov ~]$ chmod +x prog3
[aashuplecov@aashuplecov ~]$ ./prog3 -c 5
[aashuplecov@aashuplecov ~]$ ls
ls: отсутствут входные файлы
[aashuplecov@aashuplecov ~]$ ls
1.tmp  abc1      ed      'elab07018'  my_os      prog1    rsa.pub  ski.places  Изображения
2.tmp  australia ed.pub  lab07.sh     0tchet     prog2    script1  work        Музыка
3.tmp  backup    feathers lab07.sh~    outputfile prog3    script2  Видео      Общедоступные
4.tmp  bin       file.txt may         play       reports  script3  Документы  'Рабочий стол'
5.tmp  conf.txt  input.txt monthly      presentation rsa      script4  Загрузки   Шаблоны
[aashuplecov@aashuplecov ~]$ ./prog3 -r
[aashuplecov@aashuplecov ~]$ ls
abc1      ed      'elab07018'  my_os      prog1    rsa.pub  ski.places  Изображения
australia ed.pub  lab07.sh     0tchet     prog2    script1  work        Музыка
backup    feathers lab07.sh~    outputfile prog3    script2  Видео      Общедоступные
bin       file.txt may         play       reports  script3  Документы  'Рабочий стол'
conf.txt  input.txt monthly      presentation rsa      script4  Загрузки   Шаблоны
[aashuplecov@aashuplecov ~]$

```

Рис. 3.6: проверка командного файла, создающего указанное число файлов

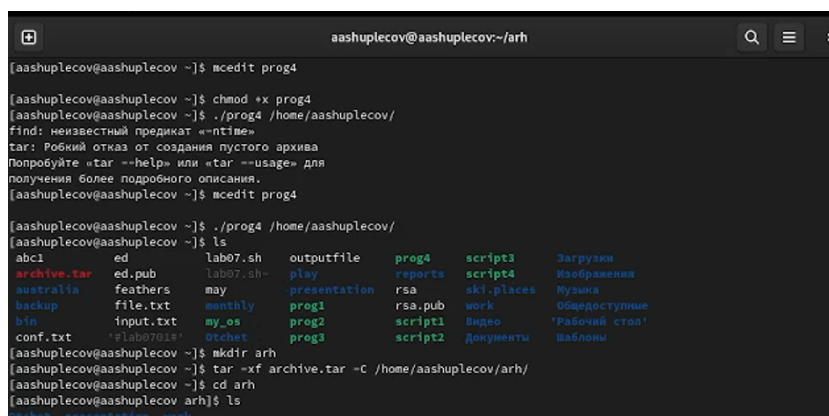
7. Напишем командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории, которые использовались менее недели тому назад.



```
prog4 [---] 22 L: [ 1+ 7 8/ 8] *(91 / 137b) 0116 0x074 [X]
while getopts ":p" opt;
do
case $opt in
p)dir="$OPTARG";;
esac
done
find $dir -mtime +0 -mtime -7 -print0 | xargs -0 tar -cf archive.tar
```

Рис. 3.7: текст командного файла, который запаковывает в архив недавно использованные файлы

8. Убедимся, что командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории, которые использовались менее недели тому назад, работает.



```
[aashuplecov@aashuplecov ~]$ mcedit prog4
[aashuplecov@aashuplecov ~]$ chmod +x prog4
[aashuplecov@aashuplecov ~]$ ./prog4 /home/aashuplecov/
find: неизвестный предмет «-mtime»
tar: Ровный отказ от создания пустого архива
Попройте «tar --help» или «tar --usage» для
получения более подробного описания.
[aashuplecov@aashuplecov ~]$ mcedit prog4
[aashuplecov@aashuplecov ~]$ ./prog4 /home/aashuplecov/
[aashuplecov@aashuplecov ~]$ ls
archive.tar  ed.pub  lab07.sh  outputfile  prog4  script3  Загрузки
australia   feathers may      presentation reports  script4  Изображения
backup      file.txt monthly prog1      rsa.pub  work    Общедоступные
bin         input.txt my_os    prog2      script1  Видео     'Рабочий стол'
conf.txt    'glab07013'  oschat  prog3      script2  Документы  Шаблоны
[aashuplecov@aashuplecov ~]$ mkdir arh
[aashuplecov@aashuplecov ~]$ tar -xf archive.tar -C /home/aashuplecov/arh/
[aashuplecov@aashuplecov ~]$ cd arh
[aashuplecov@aashuplecov arh]$ ls
oschat presentation work
```

Рис. 3.8: проверка командного файла, который запаковывает в архив недавно использованные файлы

## 4 Контрольные вопросы

### 1. Каково предназначение команды getopt?

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, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -infile_in.txt -outfile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае: 

```
while
getopts o:i:Ltr optletter do
case $optletter in
o) iflag = 1; oval =OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option
$optletter
esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равно `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый аргумент. Функция `getopts` также понимает переменные типа массив, следова-

тельно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

## 2. Какое отношение метасимволы имеют к генерации имён файлов?

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. Какие операторы управления действиями вы знаете?

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. Какие операторы используются для прерывания цикла?

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

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).

#### 6. Что означает строка `if test -f man$s/$i.$s`, встречаемая в командном файле?

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

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

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

## **Список литературы**

Кулябов Д.С. “Материалы к лабораторным работам”