

# Лабораторная работа №11

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

Рябцев И.В.; НКАбд-03-22

### 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

- Используя команды `getopts` `grep`, написать 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`.
- Написать 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.
- Написать 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).
- Написать командный файл, который с помощью команды `tag` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`).

### 3 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;

- С-оболочка (или csh) — надстройка на оболочке Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже. [1]

### 4 Выполнение лабораторной работы

- Используя команды `getopts` `grep`, написать 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;
- `-С` — различать большие и малые буквы;
- `-п` — выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом `-р`. (рис. 1-4)

```
lab11 [-M--] 47 L:[ 1+11 12/ 12] *(199 / 199b) <EOF> [*][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

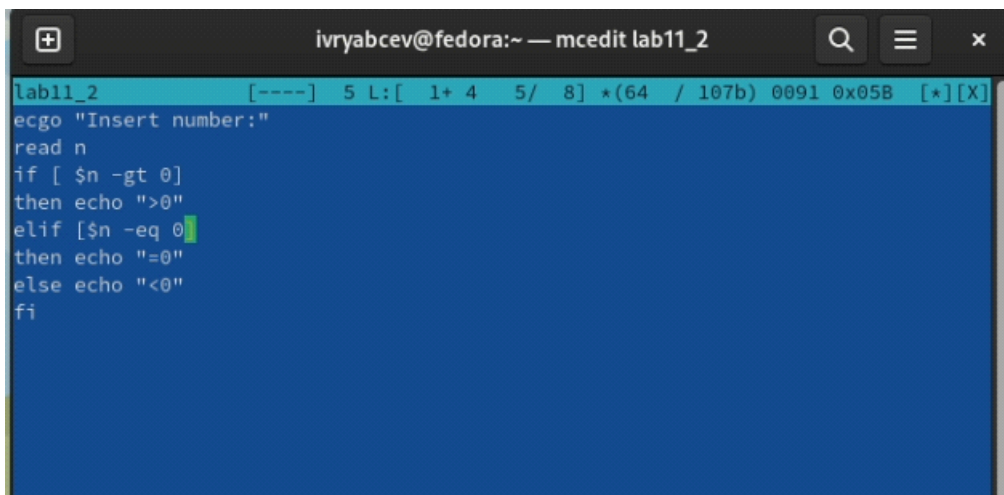
grep -n "$shablon" "$inputfile" > "$outputfile"
```

Рис. 1: Первая программа

```
[ivryabcev@fedora ~]$ ./lab11 -i lab11 -o output.txt -p n etconf -C -n
[ivryabcev@fedora ~]$ cat output.txt
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"
[ivryabcev@fedora ~]$ cl
```

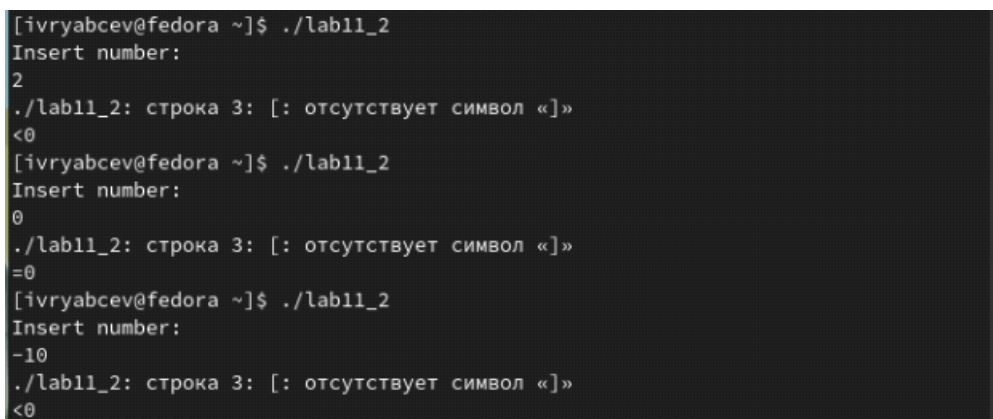
Рис. 2: Вызов программы в терминале и результат

- Написать 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. (рис. 5-7)



```
lab11_2 [-----] 5 L:[ 1+ 4 5/ 8] *(64 / 107b) 0091 0x05B [*][X]
ecgo "Insert number:"
read n
if [ $n -gt 0]
then echo ">0"
elif [ $n -eq 0]
then echo "=0"
else echo "<0"
fi
```

Рис. 3: Вторая программа



```
[ivryabcev@fedora ~]$ ./lab11_2
Insert number:
2
./lab11_2: строка 3: [: отсутствует символ «]»
<0
[ivryabcev@fedora ~]$ ./lab11_2
Insert number:
0
./lab11_2: строка 3: [: отсутствует символ «]»
=0
[ivryabcev@fedora ~]$ ./lab11_2
Insert number:
-10
./lab11_2: строка 3: [: отсутствует символ «]»
<0
```

Рис. 4: Результат

- Написать 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). (рис. 8, 9)

```
lab11_3 [-M--] 39 L: [ 1+ 4 5/ 7] *(142 / 163b) 0109 0x06D [*][X]
while getopts "c:r" opt
do
case $opt in
c)n="$OPTARG"; for i in $(seq 1 $n); do touch "$i.tnp"; done;;
r)for i in $(find -name "*.tnp"); do r $i; done;;
esac
done
```

Рис. 5: Третья программа

```
ivryabcev@fedora ~]$ ls
april          input.txt      parentdir      ski.plases
australia      ivryabcev.github.io  parentdir1    tmp
backup         '#lab07.sh#'   parentdir2    work
bin            lab07.sh       parentdir3    Видео
blog           lab11          plans         Документы
conf.txt       lab11_2        play          Загрузки
course-directory-student-template lab11_3        reports      Изображения
equiplist2     may            script01       Музыка
feathers       monthly        script02       Общедоступные
file.txt       my_os          script03       'Рабочий стол'
fun            output.txt     script04       Шаблоны
ivryabcev@fedora ~]$
```

Рис. 6: Результат

- Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). (рис. 10-12)

```
lab11_4 [-M--] 58 L: [ 1+ 7 8/ 8] *(127 / 127b) <EOF> [*][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 a
```

Рис. 7: Четвертая программа

```

[ivryabcev@fedora ~]$ ./lab11_4 /home/ivryabcev/
find: неизвестный предикат «-ntime»
tar: Робкий отказ от создания пустого архива
Попробуйте «tar --help» или «tar --usage» для
получения более подробного описания.
[ivryabcev@fedora ~]$ ls
  april          ivryabcev.github.io  parentdir1  work
  australia      '#lab07.sh#'         parentdir2  Видео
  backup         lab07.sh             parentdir3  Документы
  bin            lab11                plans       Загрузки
  blog           lab11_2              play        Изображения
  conf.txt       lab11_3              reports     Музыка
  course-directory-student-template lab11_4             script01    Общедоступные
  equiplist2     may                  script02    'Рабочий стол'
  feathers       monthly              script03    Шаблоны
  file.txt       my_os                script04
  fun            output.txt           ski.places
  input.txt      parentdir            tmp
[ivryabcev@fedora ~]$

```

Рис. 8: Результат и вывод в терминале

## 5 Выводы

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

## 6 Контрольные вопросы

- 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 -ifile_in.txt -ofile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае: `while getopts o:i:Ltr optletter do case $optletter in i) iflag=1; ival=$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 равна file\_in.txt для опции i и file\_out.doc для опции o). OPTIND является числовым индексом на упомянутый аргумент. Функция getopt 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.

- 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.

- Какие операторы управления действиями вы знаете?

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов 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.

- 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.

- 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).

- Что означает строка `if test -f mani.$s`, встречающаяся в командном файле?

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).

- 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.

## Список литературы

1. Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. Командные файлы [Электронный ресурс]. URL: <https://esystem.rudn.ru/>.