

# Лабораторная работа №11

## Дисциплина: Операционные системы

Королев Федор Константинович

### Содержание

Цель работы .....	1
Ход работы.....	1
Контрольные вопросы .....	6
Вывод.....	8

### Цель работы

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

### Ход работы

1. Используя команду `grep`, написал командный файл, который просматривает командную строку с ключами
  - `-i`inputfile - прочитать данные из указанного файла
  - `-o`outputfile - вывести данные в указанный файл
  - `-r` шаблон - указать шаблон для поиска
  - `-C` - различать большие и малые буквы
  - `-n` - выдать номер строк, а затем найти в указанном файле нужные строки, определяемые ключем `-r`.

Для данной задачи создам файл `script1.sh` (Рис. 1) и напишу соответствующий скрипт (Рис. 2):

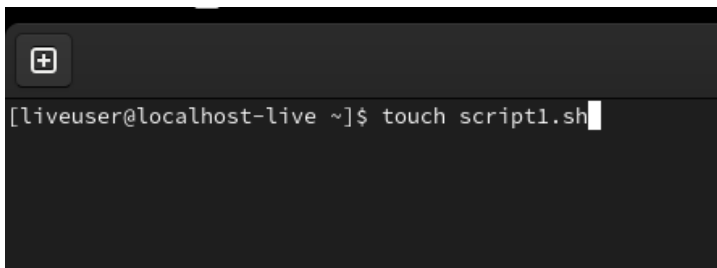


Рис. 1 создание файла script1.sh

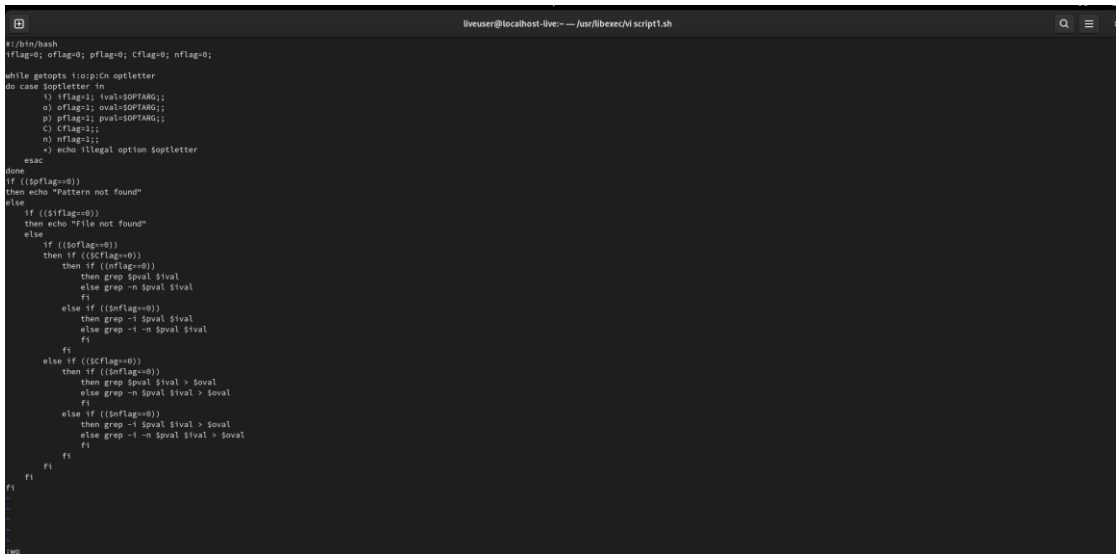


Рис. 2 код программы для первой задачи

Создадим файлы для проверки и дадим права доступа(Рис. 3):

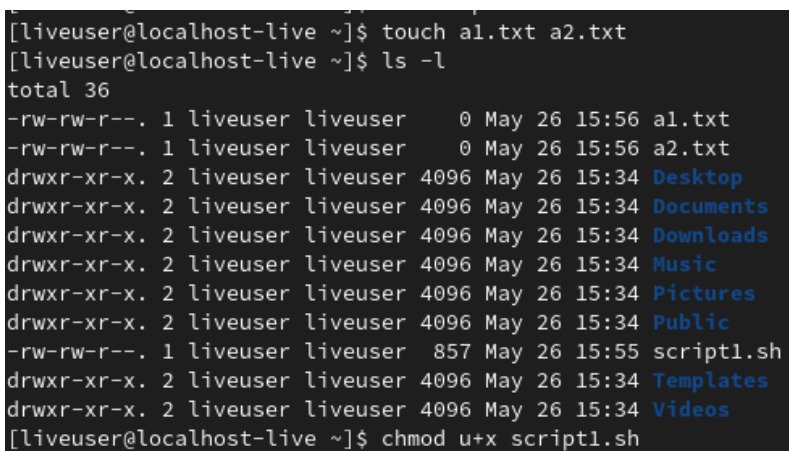


Рис. 3 создание вспомогательных файлов и предоставление прав доступа

Проверим скрипт(Рис. 4):

```

[liveuser@localhost-live ~]$ cat a1.txt
hello world
RESPECT
script1
hello hello
[liveuser@localhost-live ~]$ ./script1.sh -i a1.txt -o a2.txt -p hello -n
[liveuser@localhost-live ~]$ cat a2.txt
1:hello world
4:hello hello
[liveuser@localhost-live ~]$ ./script1.sh -i a1.txt -o a2.txt -p hello -C -n
[liveuser@localhost-live ~]$ cat a2.txt
1:hello world
4:hello hello
[liveuser@localhost-live ~]$ ./script1.sh -i a1.txt -C -n
Pattern not found
[liveuser@localhost-live ~]$ ./script1.sh -o a2.txt -p RESPECT -C -n
File not found

```

Рис. 4 проверка скрипта

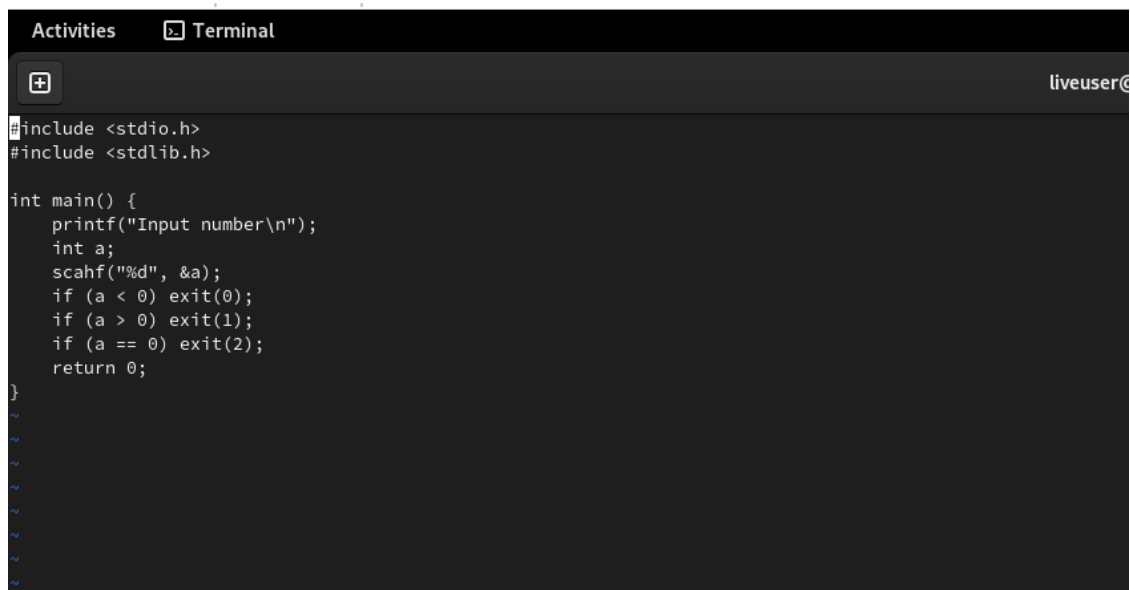
2. Напишем на Си программу(Рис. 6), которая считывает число и определяет, является ли оно большим нуля. Затем программа будет завершаться и передаст информацию в о коде завершения в оболочку. Командный файл(Рис. 7) должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено. Для данной задачи я создал файлы: number.c и number.sh(Рис. 5)

```

[liveuser@localhost-live ~]$ touch number.c
[liveuser@localhost-live ~]$ touch number.sh

```

Рис. 5 создание файлов



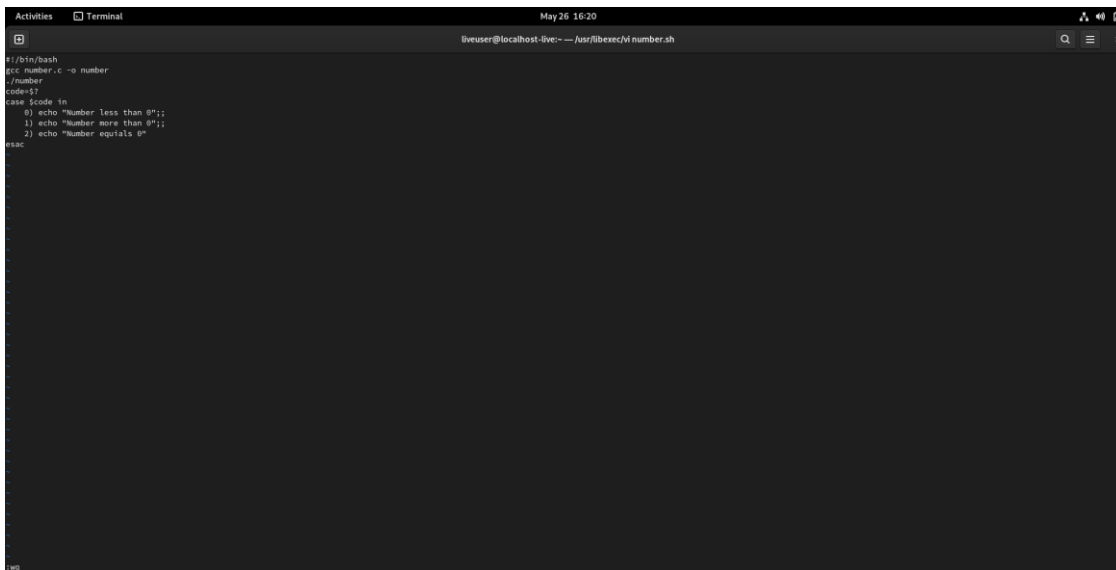
```

Activities  Terminal
+
liveuser@
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Input number\n");
    int a;
    scanf("%d", &a);
    if (a < 0) exit(0);
    if (a > 0) exit(1);
    if (a == 0) exit(2);
    return 0;
}
~
~
~
~
~
~
~

```

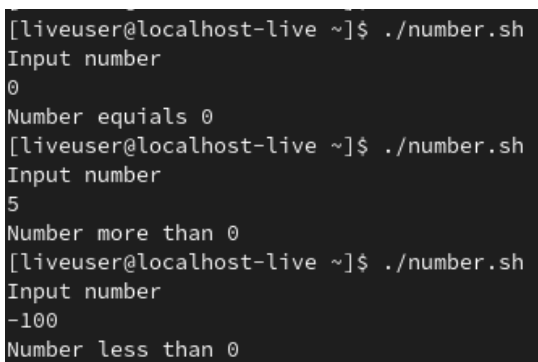
Рис. 6 программа на Си



```
Activities Terminal May 26 16:20
liveuser@localhost-live: ~ — /usr/libexec/vi number.sh
$ ./bin/bash
$ gcc number.c -o number
$ ./number
code=57
case $code in
0) echo "Number less than 0";;
1) echo "Number more than 0";;
2) echo "Number equals 0";;
*) ;;
esac
```

Рис. 7 .sh файл

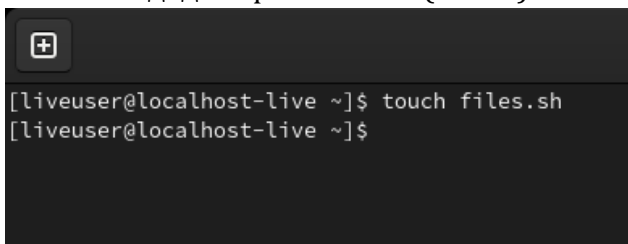
Добавим право на исполнение и проверим программу(Рис. 8):



```
[liveuser@localhost-live ~]$ ./number.sh
Input number
0
Number equals 0
[liveuser@localhost-live ~]$ ./number.sh
Input number
5
Number more than 0
[liveuser@localhost-live ~]$ ./number.sh
Input number
-100
Number less than 0
```

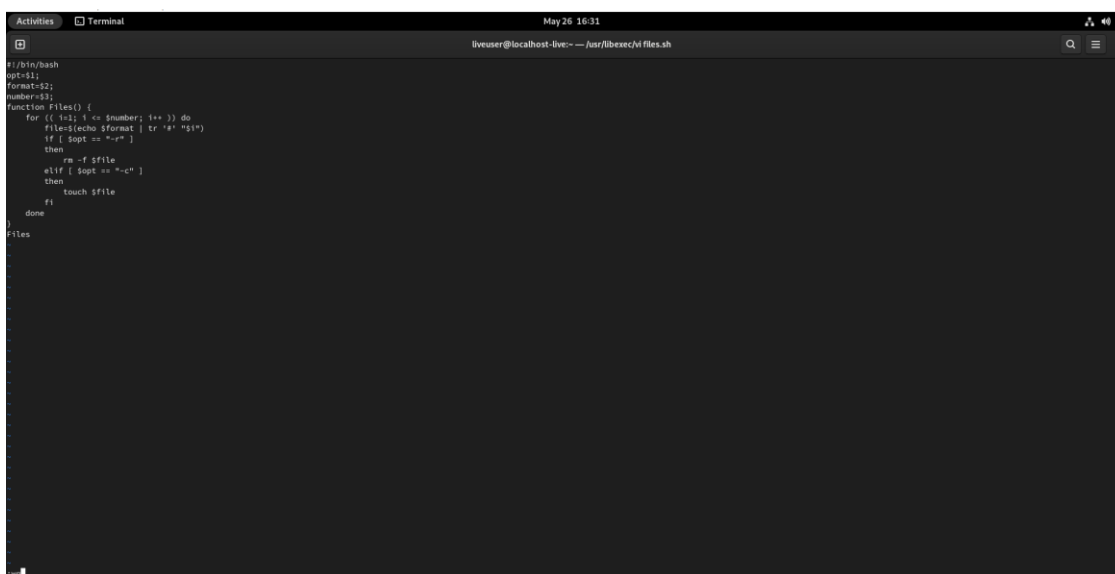
Рис. 8 проверка скрипта

3. Напишем командный файл, который будет создавать указанное число файлов пронумерованных последовательно от 1 до N. Число файлов, которые необходимо создать, передается в аргументы командной строки. Этот же командный файл должен удалять все созданные им файлы. Для данной задачи создадим файл files.sh(Рис. 9) и напишем скрипт(Рис. 10):



```
[liveuser@localhost-live ~]$ touch files.sh
[liveuser@localhost-live ~]$
```

Рис. 9 создание файлов

A terminal window titled 'Terminal' with a timestamp of 'May 26 16:31'. The prompt is 'liveuser@localhost-live:~'. The script content is as follows:

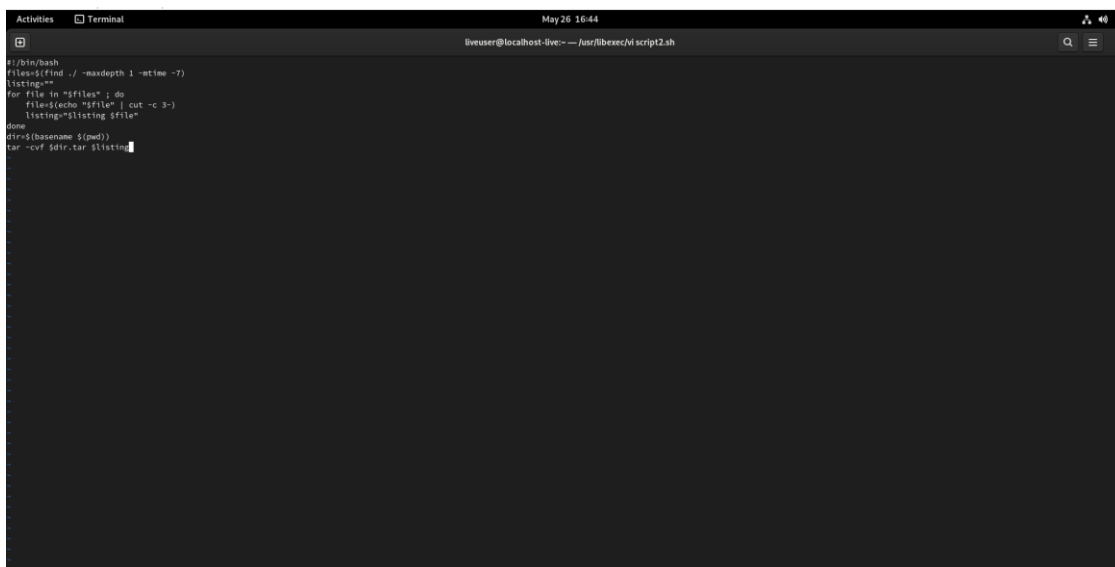
```
#!/bin/bash
opt=$1;
format=$2;
number=$3;
function Files() {
  for (( i=1; i <= $number; i++ )) do
    file=$(echo $format | tr 'a' 'i')
    if [ $opt == "-p" ]
    then
      rm -f $file
    elif [ $opt == "-c" ]
    then
      touch $file
    fi
  done
}
Files
```

Рис. 10 скрипт

4. Напишем командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад. Для данной задачи создадим файл script2.ch(Рис. 11) и напишем соответствующий скрипт(Рис. 12):

```
[liveuser@localhost-live ~]$ touch script2.sh
```

Рис. 11 создание файлов

A terminal window titled 'Terminal' with a timestamp of 'May 26 16:44'. The prompt is 'liveuser@localhost-live:~'. The script content is as follows:

```
#!/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
```

Рис. 12 написание скрипта

Добавим права доступа и проверим программу(Рис. 13):

```
[liveuser@localhost-live ~]$ ./script2.sh
.config/
.config/gtk-3.0/
.config/gtk-3.0/bookmarks
.config/dconf/
.config/dconf/user
.config/user-dirs.locale
.config/evolution/
.config/evolution/sources/
.config/evolution/sources/system-proxy.source
.config/goa-1.0/
.config/autostart/
.config/autostart/fedora-welcome.desktop
.config/gnome-initial-setup-done
.config/abrt/
.config/ibus/
.config/ibus/bus/
.config/ibus/bus/25e1461209a64f38b6fa4e94876a4cd6-unix-wayland-0
.config/pulse/
.config/pulse/cookie
.config/user-dirs.dirs
.local/
.local/share/
.local/share/gvfs-metadata/
.local/share/gvfs-metadata/home-72ecbcfd.log
.local/share/gvfs-metadata/home
.local/share/gvfs-metadata/root-ebb0fc3d.log
.local/share/gvfs-metadata/root
.local/share/icc/
.local/share/keyrings/
.local/share/flatpak/
.local/share/flatpak/db/
.local/share/sounds/
.local/share/applications/
.local/share/evolution/
.local/share/evolution/addressbook/
.local/share/evolution/addressbook/system/
.local/share/evolution/addressbook/system/photos/
.local/share/evolution/addressbook/system/contacts.db
.local/share/evolution/addressbook/trash/
.local/share/evolution/tasks/
.local/share/evolution/tasks/system/
.local/share/evolution/tasks/trash/
.local/share/evolution/calendar/
.local/share/evolution/calendar/system/
.local/share/evolution/calendar/system/calendar.ics
```

Рис. 13 проверка программы

## Контрольные вопросы

1). Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, 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 описок 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 введенные 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 getopt также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных. 2). При перечислении 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.c. 1.3. echoprogram.? –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 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, неравный 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 -fmans/i. проверяет, существует ли файл mans/i. 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.

## Вывод

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.