ЛАБОРАТОРНАЯ РАБОТА №5

**Изучение оболочки Bash ОС Linux. Сценарии**

*1. Основные сведения об оболочке Bash*. Оболочка обеспечивает взаимодействие пользователя с операционной системой. Является альтернативой графического интерфейса пользователя (GUI). Работа с оболочкой возможно в текстовом режиме. Запуск программ в оболочке эквивалентен двойному щелчку по исполняемому файлу в графическом интерфейсе. Передача аргументов в командной строке эквивалентна тому, что мы перетаскиваем что-то на иконку программы в графическом интерфейсе. Пользователь взаимодействует с оболочкой посредством командной строки. Существуют команды, способные работать только через оболочку.

В первых системах UNIX применялась оболочка *sh (shell)*. Далее было разработано несколько вариантов этой оболочки, одним из которых и является оболочка *bash*. Оболочка не выполняет никаких прикладных задач. Основная задача оболочки – обеспечение работы всех приложений и организация ввода/вывода. *Bash* также отвечает за работу с переменными окружения. Для более гибкого применения оболочка снабжена простым языком программирования, позволяющим реализовывать условны операторы, операторы цикла и некоторые другие функции. Оболочка также «умеет» передавать результаты работы одних программ другим и пользователю.

Оболочка *Bash* использует ряд символов, которые характеризуются как специальные. Вот некоторые из них:

`~! @#&: ;' "\/<>.

В зависимости от ситуации они могут трактоваться как специальные, а могут иметь буквенное значение. Применение символа «пробел» недопустимо в именах файлов.

2. *Некоторые операторы оболочки.* Рассмотрим некоторые операторы, используемые командной оболочкой.

Оператор «;» применяется для отделения одной команды от другой. Когда последние расположены на одной строке. В этом случае команды будут выполняться последовательно одна за другой:

```
admin@VirtualBox:~$ command_1_; command_2
```

Если не поставить этот оператор, то последующая команда может быть воспринята как аргумент предыдущей:

```
admin@VirtualBox:~$ command_1 command_2
```

Например, выведем сначала список запущенных процессов, а затем выполним просмотр текущего каталога:

```
admin@VirtualBox:~$ ps -f; ls -l
UID          PID  PPID  C STIME TTY          TIME CMD
admin       1883  1648  0 09:48 tty1     00:00:00 -bash
admin       1899  1883  0 09:46 tty1     00:00:00 ps -f
total 56
-rwxrwxr-x 1 admin admin 7332 сент. 23 23:21 a.out
-rw-rw-r-- 1 admin admin   78 сент. 23 23:20 hello.c
drwxrwxr-x 2 admin admin 4096 сент. 21 14:44 lab_4
```

Оператор & служит для организации команд в фоновом режиме. Если поставить этот символ после команды, оболочка вернет пользователю сразу после запуска команды, не дожидаясь, пока выполнение команды завершится:

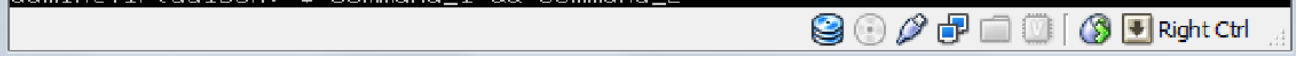
```
admin@VirtualBox:~$ ps -f & ls -l &
[1] 1907
[2] 1908
admin@VirtualBox:~$ total 56
-rwxrwxr-x 1 admin admin 7332 сент. 23 23:21 a.out
-rw-rw-r-- 1 admin admin   78 сент. 23 23:20 hello.c
drwxrwxr-x 2 admin admin 4096 сент. 21 14:44 lab_4
-rw-rw-r-- 1 admin admin    5 сент. 23 23:57 lop
-rw-rw-r-- 1 admin admin  211 сент. 23 22:55 thread.c
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Видео
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Документы
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Загрузки
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Изображения
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Музыка
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Общедоступные
drwxr-x--- 2 admin admin 4096 сент. 19 13:04 Рабочий стол
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Шаблоны
UID        PID  PPID  C STIME TTY          TIME CMD
admin     1883  1648  0 09:43 tty1     00:00:00 -bash
admin     1907  1883  0 11:12 tty1     00:00:00 ps -f

[1]-  Done                    ps -f
[2]+  Done                    ls --color=auto -l
admin@VirtualBox:~$
```

Числа в квадратных скобках определяют порядковые номера заданий.

Операторы && и || являются управляющими операторами. Если в командной строке записать

```
admin@VirtualBox:~$ command_1 && command_2
```




то команда 2 выполнится только в том случае, если выполнение команды 1 успешно завершилось. Оператор || выполнит вторую команду независимо от результатов выполнения первой:

```
admin@VirtualBox:~/lab_4$ ls -i && ls -l
183942 file.txt  183781 loop
total 8
-rw-rw-r-- 1 admin admin 13 окт.   5 01:28 file.txt
-rwxrwxrwx 1 admin admin 26 сент. 21 10:13 loop
admin@VirtualBox:~/lab_4$ ls -i ||ls -l
183942 file.txt  183781 loop
admin@VirtualBox:~/lab_4$
```

Команды в Linux выполняется следующим образом. Оболочка должна найти код команды, загрузить его в память, передать команды аргументы, заданные в командной строке, а после завершения выполнения соответствующего процесса передать каким-то образом пользователю или другому процессу результаты выполнения данной команды.

*3. Потоки ввода-вывода. Перенаправление потоков.*

Для того, чтобы записать данные в файл, процессу необходимо сначала его открыть (если он существует на диске, в противном случае файл необходимо сначала создать). При этом процесс получает *дескриптор открытого файла* – уникальное для данного процесса число, которое он будет использовать в дальнейшем. Первый открытый файл получает дескриптор 0, второй – 1 и т.д. Закончив работу с файлом, процесс должен закрыть его, освободив при этом дескриптор. Качестве файла может выступать обычный файл, файл-дырка, каналы. Поэтому далее будем подразумевать, что операции открытия, чтения, записи и закрытия осуществляются с *потоками данных*. *Дескриптор потока* – это описатель потока данных, открытого процессом.

Когда процесс (программа) запускается на выполнение, в его распоряжение предоставляется три потока (канала):

- *стандартный ввод* (*stdin*). По этому каналу данные передаются программе. Имеет дескриптор 0;
- *стандартный вывод (stdout)*. По этому каналу программа выводит результаты своей работы. Имеет дескриптор 1;
- *стандартный поток сообщений об ошибках (stderr)*. По этому каналу программы выдают информацию об ошибках. Имеет дескриптор 2.

Из стандартного потока ввода программа может только читать, а в два других – только записывать. По умолчанию входной поток связан с клавиатурой, а выходной и сообщений об ошибках с консолью. Но потоки можно перенаправлять специальными командами.

Пример потока ошибок. Умышленно зададим неверный модификатор команды *ls*:

```
admin@VirtualBox:~$ ls -
ls: cannot access -: No such file or directory
admin@VirtualBox:~$ _
```

Сообщение об ошибке появилось в командной строке.

Рассмотрим некоторые команды работы с потоками.

Команда *echo* выводит на консоль (по умолчанию) строки символов, которые были заданы ей в качестве аргументов:

```
admin@VirtualBox:~$ echo Hello!
Hello!
admin@VirtualBox:~$ _
```

Команда *cat* первоначально предназначалась для создания и просмотра содержимого файлов. В дальнейшем ее функции были расширены. Команда *cat* работает с входным и выходным потоками. По умолчанию результат команды *cat* направлен в выходной поток. Запустив *cat* без аргументов можно в этом убедиться. Сообщить команде *cat*, что мы закончили ввод можно комбинацией клавиш CTRL+D. Ввод также можно завершить комбинацией CTRL+C, сообщив оболочке, что *cat* необходимо закрыть. У *cat* есть аргументы. Если в качестве аргумента задать имя файла, то содержимое файла будет направлено во входной поток, откуда его примет команда *cat* и выдаст в выходной поток (на консоль). Т.е. этой командой можно просматривать содержимое текстовых файлов на консоли.

Обычно требуется перенаправлять потоки ввода/вывода. Для перенаправления вывода команды в файл служит оператор «>». Например, сохраним в файл *ls_out* (который поместим в каталоге *lab_5*) результат выполнения команды *ls –l* и сразу просмотрим его содержимое:

```
admin@VirtualBox:~$ ls -l > /home/admin/lab_5/ls_out && cat /home/admin/lab_5/ls
_out
total 44
-rw-rw-r-- 1 admin admin   14 сент. 25 09:38 echo_out
drwxrwxr-x 2 admin admin 4096 окт.  19 09:18 lab_4
drwxrwxr-x 2 admin admin 4096 окт.  19 09:32 lab_5
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Видео
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Документы
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Загрузки
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Изображения
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Музыка
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Общедоступные
drwxr-x--- 2 admin admin 4096 сент. 19 13:04 Рабочий стол
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Шаблоны
```

Расширение фала *ls_out* можно не указывать. (Мы знаем, что расширение файла в Linux носит весьма условный характер). В данном случае выходной поток команды *ls –l* был перенаправлен из консоли в файл.

Перенаправим выходной поток команды *echo* в файл с именем *echo_out*:

```
admin@VirtualBox:~$ echo Hello, world! >echo_out
admin@VirtualBox:~$ cat echo_out
Hello, world!
admin@VirtualBox:~$ _
```

В результате создался файл *echo_out,* в котором содержится текстовая строка «*Hello, world!*».

Оператор «>>» позволяет перенаправлять вывод в существующий файл. При этом содержимое файла не удалится, а новая порция информации допишется в конец файла. Добавим в файл *ls_out* информацию о индексных дескрипторах объектов файловой системы, расположенных в текущем каталоге:

```
admin@VirtualBox:~$ ls -i >> /home/admin/lab_5/ls_out && cat /home/admin/lab_5/l
s_out
total 44
-rw-rw-r-- 1 admin admin   14 сент. 25 09:38 echo_out
drwxrwxr-x 2 admin admin 4096 окт.  19 09:18 lab_4
drwxrwxr-x 2 admin admin 4096 окт.  19 09:32 lab_5
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Видео
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Документы
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Загрузки
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Изображения
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Музыка
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Общедоступные
drwxr-x--- 2 admin admin 4096 сент. 19 13:04 Рабочий стол
drwxr-xr-x 2 admin admin 4096 сент. 19 13:04 Шаблоны
183941 echo_out
183777 lab_4
183942 lab_5
131486 Видео
131473 Документы
131438 Загрузки
131485 Изображения
131484 Музыка
131472 Общедоступные
131432 Рабочий стол
131447 Шаблоны
admin@VirtualBox:~$
```

Можно также перенаправлять и входные потоки с помощью оператора «<». По умолчанию стандартный ввод осуществляется с клавиатуры. Для примера используем команду *wc –m <введенные символы>*, которая ведет подсчет количества символов, введенных с клавиатуры:

```
admin@VirtualBox:~/lab_5$ wc -m < echo_out
14
admin@VirtualBox:~/lab_5$
```

Мы перенаправили входной поток: вместо клавиатуры указали файл на диске

Ещё одним полезным оператором перенаправления вывода является оператор «|». Он служит для организации программного канала (*конвейера*). Например, при использовании двух команд, в которых результаты работы одной из них служат входными данными для другой, можно использовать этот оператор, поместив его между командами. При этом стандартный выходной поток команды, стоящей слева от символа «|» перенаправляется на стандартный входной поток программы, расположенной справа от символа. Например, выполним команды *cat echo_out | wc –w*:

Выходные результаты команды *cat* перенаправляются на вход команды *wc –w*. В результате на консоль будет выведено количество слов в файле *echo_out*. С помощью данного оператора

```
admin@VirtualBox:~/lab_5$ cat echo_out |wc -w
2
```

можно объединять много команд в одну цепочку. При этом оболочка одновременно вызывает на выполнение все команды, помещенные в конвейер, запуская отдельный экземпляр оболочки для каждой команды. Как только первая команда выдает результат в свой выходной поток, вторая команда начинает его обрабатывать. Если необходимо, чтобы команда полностью завершилась до начала выполнения следующей команды, необходимо в конвейере использовать оператор «;». Перед каждой точкой с запятой оболочка будет останавливаться и ожидать, пока завершится выполнение всех предыдущих команд, включенных в конвейер.

*4. Параметры и переменные оболочки.* Все параметры разделяются на три класса: *позиционные параметры, специальные параметры* и *переменные оболочки.*

Имена *позиционных параметров* состоят из одной или более цифр (кроме одиночного нуля). Значениями позиционных параметров являются аргументы, которые были заданы при запуске оболочки (первый аргумент является значением позиционного параметра 1, и т.д.) Изменить значение позиционного параметра можно с помощью команды *set.*

*Специальные параметры* являются шаблонами, замена которых производится согласно таблице ниже.

| Параметр | Правила замены |
|---|---|
| * | Заменяется позиционными параметрами, начиная с первого. Если замена производится внутри двойных кавычек, то этот параметр заменяется на одно единственное слово, составленное из всех позиционных параметров, разделенных первым символом специальной переменной IFS |
| @ | Заменяется позиционными параметрами, начиная с первого. Если замена производится внутри двойных кавычек, то каждый параметр заменяется отдельным словом |
| # | Заменяется десятичным значением числа позиционных параметров |
| ? | Заменяется статусом входа последнего из выполнявшихся на переднем плане программных каналов |
| $ | Заменяется идентификатором процесса (PID) оболочки |

Присваивать значения специальным параметрам нельзя! На них можно только ссылаться.

*Переменная* с точки зрения оболочки – это параметр, обозначенный именем.

Именем параметра может быть слово, состоящее из алфавитных символов, цифр и знаков подчеркивания. Параметр считается установленным, если ему присвоено какое-то значение. Для вывода значения параметра необходимо использовать символ $.

Присвоим переменной *num* значение 5 и выведем его на экран:

```
admin@VirtualBox:~/lab_5$ num=5
admin@VirtualBox:~/lab_5$ echo num = $num
num = 5
```

Присвоим переменной s*tring* значение *World* и выведем его на экран:

```
admin@VirtualBox:~/lab_5$ string=World; echo $string
World
admin@VirtualBox:~/lab_5$
```

Оболочка имеет свои собственные переменные. Значения некоторых переменных пользователь может изменять. Например, переменная *PS1* задает вид приглашения, которое оболочка *Bash* выводит, когда ожидает ввода данных или команды от пользователя. Просмотрим значение этой переменной:

```
admin@VirtualBox:~/lab_5$ echo $PS1
${debian_chroot:+($debian_chroot)}\u@\h:\w\$
admin@VirtualBox:~/lab_5$
```

Переменная *PS1* содержит специальные символы. В таблице ниже приведены некоторые из них.

| Символ | Значение символа |
|---|---|
| \a | Звуковой сигнал |
| \d | Дата в формате ДД,ММ,ЧЧ |
| \h | Имя хоста до первой точки |
| \u | Имя пользователя, запустившего оболочку |
| \w | Полное имя текущего рабочего каталога (от корня) |
| \# | Текущий номер команды |
| \s | Имя оболочки |
| \n | Новая строка (перевод строки) |
| \@ | Текущее время в 12-ти часовом формате |
| \$ | Символ #, если оболочка запущена администратором, и символ $, если оболочка запущена обычным пользователем |

Изменим значение переменной *PS1*– отобразим имя пользователя, запустившего оболочку, имя хоста, имя рабочего каталога и текущие дату и время:

```
admin@VirtualBox ~/lab_5Вс. нояб. 01 02:51 PS1="\u@\h\w \d \@"
admin@VirtualBox~/lab_5 Вс. нояб. 01 02:52
```

*Переменная PATH* задает перечень путей к каталогам, в которых оболочка осуществляет поиск файлов в тех случаях, когда полный путь к файлу не задан в командной строке. Отдельные каталоги в этом перечне отделяются двоеточиями. По умолчанию данная переменная содержит пути к следующим каталогам:

```
admin@VirtualBox:~/lab_5$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games
admin@VirtualBox:~/lab_5$
```

*Переменная IFS* задает разделители полей, которые используются при операции разделения слов при преобразованиях командной строки, выполняемых оболочкой перед тем, как запустить командную строку на исполнение.

*Переменная PWD* содержит путь текущего каталога. Значение этой переменной изменяется при каждом запуске команды *cd*.

```
admin@VirtualBox:~/lab_5$ echo $PWD
/home/admin/lab_5
admin@VirtualBox:~/lab_5$
```

*Переменная HOME* содержит полное имя домашнего каталога пользователя:

```
admin@VirtualBox:~/lab_5$ echo $HOME
/home/admin
admin@VirtualBox:~/lab_5$
```

*5. Скрипты оболочки. Скрипт* (сценарий) оболочки – это просто файл, содержащий последовательность команд оболочки. Команды аналогичны тем, которые пользователь вводит непосредственно в командную строку. При запуске скрипта запускается новый процесс. Простейший скрипт мы уже создавали в лабораторной работе №4 – это файл бесконечного цикла *loop*.

Для запуска сценария в текущей сессии существует команда *source <имя файла>[аргументы]* или ее синоним . *<имя файла>[аргументы]*. Данная команда читает и выполняет все команды из файла скрипта. Например, создадим скрипт, выводящий на экран строку *«Hello, world!»* и запустим его:

```
admin@VirtualBox:~/lab_5$ cat >script
echo Hello, world!
admin@VirtualBox:~/lab_5$ . /home/admin/lab_5/script
Hello, world!
admin@VirtualBox:~/lab_5$ _
```

При запуске скрипта необходимо указывать полное имя файла, т.е. путь и собственное имя. Если добавить в переменную PATH путь к скрипту, то его запуск можно осуществлять только по собственному имени файла.

Запуск скрипта возможен и с помощью нового экземпляра оболочки *bash*:

```
admin@VirtualBox:~/lab_5$ bash /home/admin/lab_5/script
Hello, world!
admin@VirtualBox:~/lab_5$
```

Обычно первая строка в скрипте указывает на программу, которая будет его выполнять. В нашем случае, команды выполняет оболочка *bash*, поэтому будет нелишним добавить в скрипт такую строчку #! */bin/bash:*

```
admin@VirtualBox:~/lab_5$ cat > script
#! /bin/bash
echo Hello, world!
admin@VirtualBox:~/lab_5$ . /home/admin/lab_5/script
Hello, world!
```

Сценарий может принимать параметры. Ниже приведен пример сценария с тремя параметрами:

```
admin@VirtualBox:~/lab_5$ cat >script_param
echo Первым параметром является $1
echo вторым параметром является $2
echo Третьим параметром является $3
admin@VirtualBox:~/lab_5$ . /home/admin/lab_5/script_param one two three
Первым параметром является one
вторым параметром является two
Третьим параметром является three
admin@VirtualBox:~/lab_5$ _
```

Количество параметров может быть любым. В качестве параметров также можно указывать имена файлов и каталогов. Например, создадим сценарий, который создает в рабочем каталоге каталог, имя которого необходимо ввести с клавиатуры, а затем выводит содержимое текущего каталога на экран:

```
admin@VirtualBox:~/lab_5$ cat >script_dir
#! /bin/bash
mkdir $1
ls -li
admin@VirtualBox:~/lab_5$ . /home/admin/lab_5/script_dir dir
total 48
184093 -rw-rw-r-- 1 admin admin   19 нояб.  1 17:47 count
184101 drwxrwxr-x 2 admin admin 4096 нояб. 23 09:28 dir
131426 -rw-rw-r-- 1 admin admin   14 окт.  19 09:39 echo_out
184086 -rw-rw-r-- 1 admin admin   49 нояб.  1 17:46 hot.txt
183912 -rw-rw-r-- 1 admin admin  116 окт.  19 10:53 ls_out
184084 -rw-rw-r-- 1 admin admin    2 нояб.  1 12:28 num
184005 -rw-rw-r-- 1 admin admin   84 окт.  31 22:00 ps
184096 -rw-rw-r-- 1 admin admin   32 нояб. 23 08:53 script
184094 -rw-rw-r-- 1 admin admin   59 нояб. 23 09:01 script_arg
184100 -rw-rw-r-- 1 admin admin   29 нояб. 23 09:28 script_dir
184098 -rw-rw-r-- 1 admin admin  179 нояб. 23 09:20 script_param
184030 -rw-rw-r-- 1 admin admin 1967 нояб.  1 17:10 variables
```

# ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

0. Подробно изучите материал, представленный в данной работе.
1. Создайте в домашнем каталоге каталог *lab_5*. Все дальнейшие действия выполняйте в этом каталоге.

1.1. Выполните две любые команды (например, *ls* и *ps*), разделяя их оператором «;».

1.2. Запустите несколько команд в фоновом режиме. Как узнать, что выполнение каждой команды закончено?

1.3. Перенаправьте вывод команды *ls -li* в файл с именем <*ls_out*>. Расширение файла не указывать. Просмотрите его содержимое.

1.4. Добавьте в файл <*ls_out*> информацию о текущем каталоге и о всех запущенных процессах пользователя.

1.5. Составьте конвейер, который выполняет следующие действия: подсчитывает количество символов в указанном файле (файл необходимо предварительно создать).

2. Работа с переменными оболочки

2.1. Создайте три переменных с именами *var1, var2, var3* и присвойте им произвольные значения (числовые или текстовые).

2.2. Вывести на экран значения переменных *var1* и *var2*, а значение переменной *var3* сохранить в файл с именем *var_value*. Просмотрите содержимое файла *var_value.*

2.3. Измените приглашение командной строки следующим образом:
*<имя пользователя>@<имя хоста до первой точки>:<имя оболочки> <звуковой сигнал>*.

3. Работа со скриптами (файлы сценариев создавать в каталоге *lab_5*)

3.1. Создайте файл сценария без параметров, выполняющий следующие действия:

- выводит на экран любую текстовую строку;
- выводит на экран содержимое каталога *lab_5.*

3.2. Создайте файл сценария с тремя параметрами, выполняющий действия в указанном порядке:

- создает каталог *dir1* в текущем каталоге;
- создает в каталоге *dir1* три пустых файла с произвольными именами;
- сохраняет результат выполнения команды *ls –l* для каталога *dir1* в файл с именем *script_out* (файл *script_out* должен размещаться в каталоге *lab_5*);
- закрывает доступ к файлу *script_out* для всех, кроме его владельца;
- создает переменную с именем *var* и выводит ее содержимое на экран.

Имена каталога *dir1,* файла *script_out* и переменной *var* должны выступать в качестве параметров скрипта.

4. Подготовьте ответы на вопросы:

4.1. Что такое оболочка *Bash*? Какие специальные символы она использует?

4.2. Объясните использование следующих операторов: «;», «&», «&&», «||».

4.3. Что называют дескриптором открытого файла?

4.4. Что называют потоками ввода-вывода? Какие стандартные потоки вы знаете?

4.5. Что такое дескриптор потока?

4.6. Поясните действия операторов «>», «<», «>>», и «|».

4.7. Что такое файл сценария?

5. Будьте готовы ответить на дополнительные вопросы преподавателя по данной теме.