

# **Отчёт по лабораторной работе №14**

**Дисциплина: Операционные системы**

Верниковская Екатерина Андреевна

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Ответы на контрольные вопросы	16
5	Выводы	19
6	Список литературы	20

## Список иллюстраций

3.1	Создание файла lab14_1.sh и добавление прав на исполнение . . .	8
3.2	Написанная программа для lab14_1.sh . . . . .	8
3.3	Проверка работы командного файла lab14_1.sh . . . . .	9
3.4	Команда . . . . .	10
3.5	Содержимое каталога /usr/share/man/man1 . . . . .	10
3.6	Создание файла lab14_2.sh и добавление прав на исполнение . . .	10
3.7	Написанная программа для lab14_2.sh . . . . .	11
3.8	man команды ls (работа командного файла lab14_2.sh) . . . . .	12
3.9	man команды chmod (работа командного файла lab14_2.sh) . . . .	12
3.10	man команды cd (работа командного файла lab14_2.sh) . . . . .	13
3.11	Проверка работы командного файла lab14_2.sh . . . . .	13
3.12	Создание файла lab13_3.sh и добавление прав на исполнение . . .	13
3.13	Написанная программа для lab14_3.sh . . . . .	14
3.14	Проверка работы командного файла lab14_3.sh . . . . .	15

## Список таблиц

# 1 Цель работы

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

## 2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени  $t_1$  дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени  $t_2 < t_1$ , также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в

диапазоне от 0 до 32767.

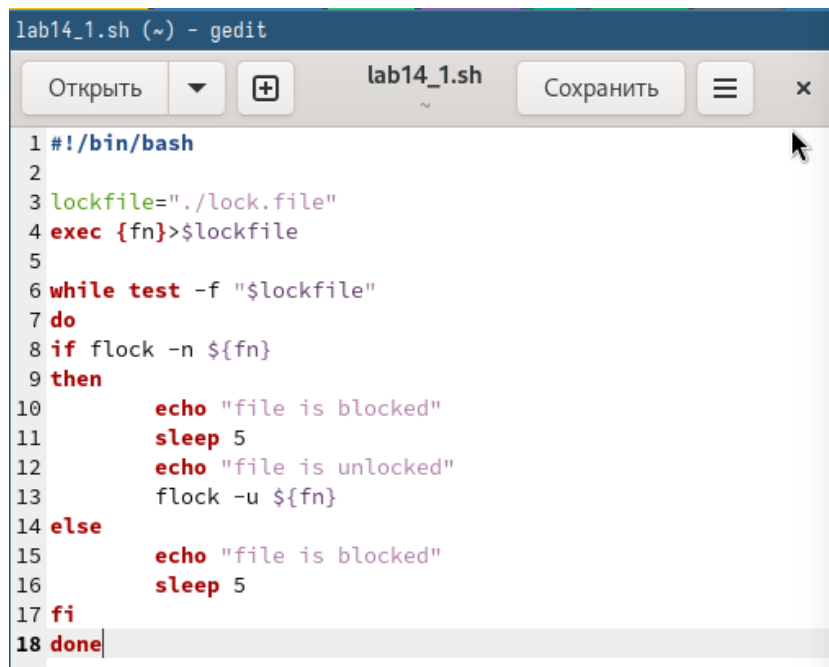
### 3 Выполнение лабораторной работы

Создаю файл для первого задания с расширением sh и делаю его исполняемым (рис. 3.1)

```
[eavernikovskaya@eavernikovskaya ~]$ touch lab14_1.sh
[eavernikovskaya@eavernikovskaya ~]$ chmod +x lab14_1.sh
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.1: Создание файла lab14\_1.sh и добавление прав на исполнение

Открываю файл lab14\_1.sh в текстовом редакторе gedit и пишу командный файл, реализующий упрощённый механизм семафоров (подробнее см. в задании №1) (рис. 3.2)



```
lab14_1.sh (~) - gedit
Открыть  lab14_1.sh  Сохранить
1 #!/bin/bash
2
3 lockfile="./lock.file"
4 exec {fn}>$lockfile
5
6 while test -f "$lockfile"
7 do
8   if flock -n ${fn}
9   then
10     echo "file is blocked"
11     sleep 5
12     echo "file is unlocked"
13     flock -u ${fn}
14   else
15     echo "file is blocked"
16     sleep 5
17   fi
18 done
```

Рис. 3.2: Написанная программа для lab14\_1.sh



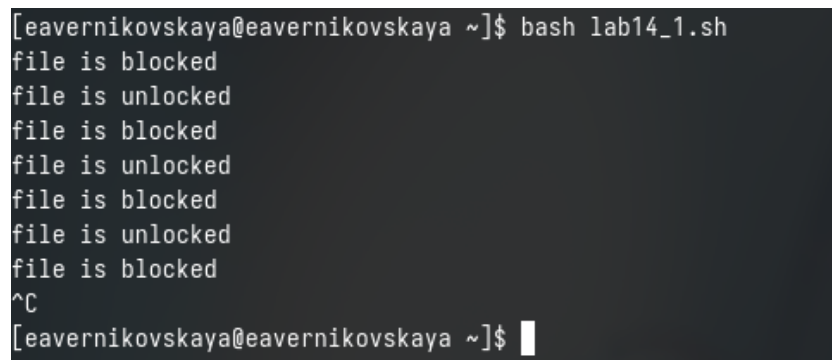
Программа для задания №1:

```
#!/bin/bash

lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
    if flock -n ${fn}
    then
        echo "file is blocked"
        sleep 5
        echo "file is unlocked"
        flock -u ${fn}
    else
        echo "file is blocked"
        sleep 5
    fi
done
```

Далее запускаю файл с помощью bash и проверяю работу командного файла (рис. 3.3)



```
[eavernikovskaya@eavernikovskaya ~]$ bash lab14_1.sh
file is blocked
file is unlocked
file is blocked
file is unlocked
file is blocked
file is unlocked
file is blocked
^C
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.3: Проверка работы командного файла lab14\_1.sh

Изучаю содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд (рис. 3.4), (рис. 3.5)

```
[eavernikovskaya@eavernikovskaya ~]$ ls /usr/share/man/man1/
```

Рис. 3.4: Команда

```
mtx-metapost.1.gz  zcmp.1.gz
mtx-modules.1.gz  zdiff.1.gz
mtx-package.1.gz  zenity.1.gz
mtx-patterns.1.gz zforce.1.gz
mtx-pdf.1.gz      zgrep.1.gz
mtx-plain.1.gz    zip.1.gz
mtx-profile.1.gz  zipcloak.1.gz
mtx-rsync.1.gz    zipdetails.1.gz
mtxrun.1.gz       zipgrep.1.gz
mtx-scite.1.gz    zipinfo.1.gz
mtx-server.1.gz   zipnote.1.gz
mtx-spell.1.gz    zipsplit.1.gz
mtx-texworks.1.gz zless.1.gz
mtx-timing.1.gz   zmore.1.gz
mtx-tools.1.gz    znew.1.gz
mtx-unicode.1.gz  zsoelim.1.gz
mtx-unzip.1.gz    zvbi-atsc-cc.1.gz
mtx-update.1.gz   zvbi-chains.1.gz
mtx-vscode.1.gz   zvbi-ntsc-cc.1.gz
mtx-watch.1.gz
```

Рис. 3.5: Содержимое каталога `/usr/share/man/man1`

Создаю файл для второго задания с расширением `sh` и делаю его исполняемым (рис. 3.6)

```
[eavernikovskaya@eavernikovskaya ~]$ touch lab14_2.sh
[eavernikovskaya@eavernikovskaya ~]$ chmod +x lab14_2.sh
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.6: Создание файла `lab14_2.sh` и добавление прав на исполнение

Открываю файл `lab14_2.sh` в текстовом редакторе `gedit` и пишу командный файл, который будет реализовывать команду `man` (рис. 3.7)



Рис. 3.7: Написанная программа для lab14\_2.sh

Программа для задания №2:

```
#!/bin/bash
```

```
a=$1
```

```
if test -f "/usr/share/man/man1/$a.1.gz"
```

```
then less /usr/share/man/man1/$a.1.gz
```

```
else
```

```
    echo "there is no such command"
```

```
fi
```

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.8), (рис. 3.9), (рис. 3.10), (рис. 3.11)

```
foot
ESC[4mL$ESC[24m(1)                                User Commands
ESC[4mL$ESC[24m(1)

ESC[1mNAMEESC[0m
ls - list directory contents

ESC[1mSYNOPSISESC[0m
ESC[1m$ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4mFILEESC[24m]...

ESC[1mDESCRIPTIONESC[0m
List information about the FILEs (the current directory by default). Sort entries alphabetically if none of ESC[1m-cft
u-ESC[0m
ESC[1mvSUX ESC[22mnor ESC[1m--sort ESC[22mis specified.

Mandatory arguments to long options are mandatory for short options too.

ESC[1m-aESC[22m, ESC[1m--allESC[0m
do not ignore entries starting with .

ESC[1m-AESC[22m, ESC[1m--almost-allESC[0m
do not list implied . and ..

ESC[1m-authorESC[0m
with ESC[1m-lESC[22m, print the author of each file

ESC[1m-bESC[22m, ESC[1m--escapeESC[0m
print C-style escapes for nongraphic characters

ESC[1m-block-sizeESC[22m=ESC[4mSIZEESC[0m
with ESC[1m-lESC[22m, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

ESC[1m-BESC[22m, ESC[1m--ignore-backupsESC[0m
do not list implied entries ending with ~

/usr/share/man/man1/ls.1.gz
```

Рис. 3.8: man команды ls (работа командного файла lab14\_2.sh)

```
foot
ESC[4mCHMOD$ESC[24m(1)                                User Commands
ESC[4mCHMOD$ESC[24m(1)

ESC[1mNAMEESC[0m
chmod - change file mode bits

ESC[1mSYNOPSISESC[0m
ESC[1mchmod ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mMODEESC[24m[ESC[4m,MODEESC[24m]... ESC[4mFILEESC[24m]...
ESC[1mchmod ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mOCTAL-MODEESC[24m ESC[4mFILEESC[24m]...
ESC[1mchmod ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4m--reference=RFILEESC[24m ESC[4mFILEESC[24m]...

ESC[1mDESCRIPTIONESC[0m
This manual page documents the GNU version of ESC[1mchmodESC[22m. ESC[1mchmod ESC[22mchanges the file mode bits of each
given file according
to ESC[4mmodeESC[24m, which can be either a symbolic representation of changes to make, or an octal number representing
the bit
pattern for the new mode bits.

The format of a symbolic mode is [ESC[1mugoaESC[22m...][ESC[1m--=ESC[22m][ESC[4mpermsESC[24m...], where ESC[4mperm
ESC[24m is either zero or more letters from the
set ESC[1mrwxXstESC[22m, or a single letter from the set ESC[1mugoaESC[22m. Multiple symbolic modes can be given, separated
by commas.

A combination of the letters ESC[1mugoaESC[22mcontrols which users' access to the file will be changed: the user who
owns it
(ESC[1muESC[22m), other users in the file's group (ESC[1mgESC[22m), other users not in the file's group (ESC[1moESC[22m),
or all users (ESC[1maESC[22m). If none of
these are given, the effect is as if (ESC[1maESC[22m) were given, but bits that are set in the umask are not affected.

The operator ESC[1m+ESC[22mcauses the selected file mode bits to be added to the existing file mode bits of each file;
ESC[1m-ESC[22mcauses
them to be removed; and ESC[1m-=ESC[22mcauses them to be added and causes unmentioned bits to be removed except that
a direc-
tory's unmentioned set user and group ID bits are not affected.

/usr/share/man/man1/chmod.1.gz
```

Рис. 3.9: man команды chmod (работа командного файла lab14\_2.sh)

```
foot
ESC[4mBASH_BUILTINSESC[24m(1)          General Commands Manual          ESC[4mB
ASH_BUILTINSESC[24m(1)

ESC[1mNAMEESC[0m
: , .. [, alias, bg, bind, break, builtin, caller, cd, command, compgen, complete, compopt, continue, declare, dirs,
disown, echo, enable, eval, exec, exit, export, false, fc, fg, getopts, hash, help, history, jobs, kill, let, local,
logout, mapfile, popd, printf, pushd, pwd, read, readarray, readonly, return, set, shift, shopt, source, suspend,
test, times, trap, true, type, typeset, ulimit, umask, unalias, unset, wait - bash built-in commands, see ESC[1mbashESC[
22m(1)

ESC[1mBASH BUILTIN COMMANDSESC[0m
Unless otherwise noted, each builtin command documented in this section as accepting options preceded by ESC[1m- ESC[2
2maccepts
ESC[1m-- ESC[22mto signify the end of the options. The ESC[1m-ESC[22m, ESC[1mtrueESC[22m, ESC[1mfalseESC[22m, and ESC[1
ntestESC[22m/ESC[1m-ESC[22mbuiltins do not accept options and do not
treat ESC[1m-- ESC[22mspecially. The ESC[1mexitESC[22m, ESC[1mlogoutESC[22m, ESC[1mreturnESC[22m, ESC[1mbreakESC[22m,
ESC[1mcontinueESC[22m, ESC[1mletESC[22m, and ESC[1mshiftESC[22mbuiltins accept and process arguments
beginning with ESC[1m- ESC[22mwithout requiring ESC[1m-ESC[22m. Other builtins that accept arguments but are not spec
ified as accepting op
tions interpret arguments beginning with ESC[1m- ESC[22mas invalid options and require ESC[1m-- ESC[22mto prevent this i
nterpretation.
ESC[1m- ESC[22mESC[4margumentsESC[24m]
No effect; the command does nothing beyond expanding ESC[4margumentsESC[24m and performing any specified redi
rections.
The return status is zero.

ESC[1m- ESC[4mESC[22mfilenameESC[24m ESC[4margumentsESC[24m]
ESC[1msource ESC[4mESC[22mfilenameESC[24m ESC[4margumentsESC[24m]
Read and execute commands from ESC[4mfilenameESC[24m in the current shell environment and return the exit stat
us of the
last command executed from ESC[4mfilenameESC[24m. If ESC[4mfilenameESC[24m does not contain a slash, filenames i
n ESC[1mPATH ESC[22mare used to
find the directory containing ESC[4mfilenameESC[24m, but ESC[4mfilenameESC[24m does not need to be executable.
The file searched for
/usr/share/man/man1/cd.1.gz
```

Рис. 3.10: man команды cd (работа командного файла lab14\_2.sh)

```
eavernikovskaya@eavernikovskaya ~]$ ./lab14_2.sh ls
eavernikovskaya@eavernikovskaya ~]$ ./lab14_2.sh chmod
eavernikovskaya@eavernikovskaya ~]$ ./lab14_2.sh cd
eavernikovskaya@eavernikovskaya ~]$
```

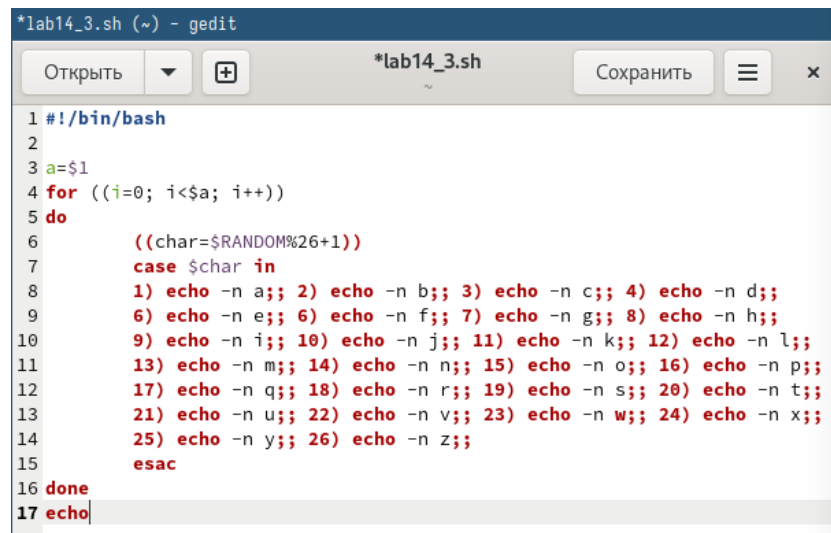
Рис. 3.11: Проверка работы командного файла lab14\_2.sh

Создаю файл для третьего задания с расширением sh и делаю его исполняе-  
мым (рис. 3.12)

```
eavernikovskaya@eavernikovskaya ~]$ touch lab14_3.sh
eavernikovskaya@eavernikovskaya ~]$ chmod +x lab14_3.sh
eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.12: Создание файла lab13\_3.sh и добавление прав на исполнение

Открываю файл lab14\_3.sh в текстовом редакторе gedit и пишу командный  
файл, генерирующий случайную последовательность букв латинского алфави-  
та (рис. 3.13)



```
*lab14_3.sh (~) - gedit
Открыть *lab14_3.sh Сохранить x
1 #!/bin/bash
2
3 a=$1
4 for ((i=0; i<$a; i++))
5 do
6     ((char=$RANDOM%26+1))
7     case $char in
8         1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;;
9         6) echo -n e;; 6) echo -n f;; 7) echo -n g;; 8) echo -n h;;
10        9) echo -n i;; 10) echo -n j;; 11) echo -n k;; 12) echo -n l;;
11        13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;;
12        17) echo -n q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;;
13        21) echo -n u;; 22) echo -n v;; 23) echo -n w;; 24) echo -n x;;
14        25) echo -n y;; 26) echo -n z;;
15    esac
16 done
17 echo
```

Рис. 3.13: Написанная программа для lab14\_3.sh

Программа для задания №3:

```
#!/bin/bash
```

```
a=$1
```

```
for ((i=0; i<$a; i++))
```

```
do
```

```
    ((char=$RANDOM%26+1))
```

```
    case $char in
```

```
        1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;;
```

```
        6) echo -n e;; 6) echo -n f;; 7) echo -n g;; 8) echo -n h;;
```

```
        9) echo -n i;; 10) echo -n j;; 11) echo -n k;; 12) echo -n l;;
```

```
        13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;;
```

```
        17) echo -n q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;;
```

```
        21) echo -n u;; 22) echo -n v;; 23) echo -n w;; 24) echo -n x;;
```

```
        25) echo -n y;; 26) echo -n z;;
```

```
    esac
```

```
done
```

```
echo
```

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.14)

```
[eavernikovskaya@eavernikovskaya ~]$ bash lab14_3.sh 13  
irmsbepsqzixy  
[eavernikovskaya@eavernikovskaya ~]$ bash lab14_3.sh 100  
ahzszadpjilyvyayzebhlmslnkxrtwxhiipyycybnyncoopktocrclrxoannipzhmkruexzqjndosyzqrqzvkkycgvi  
[eavernikovskaya@eavernikovskaya ~]$ bash lab14_3.sh 3  
zpb  
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.14: Проверка работы командного файла lab14\_3.sh

## 4 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

```
while [$1 != "exit"]
```

В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [ и перед второй скобкой ] выражение \$1 необходимо взять в “”, потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так: `while [ “$1” != “exit” ]`

2. Как объединить (конкатенация) несколько строк в одну?

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: - Первый: `VAR1=“Hello,” VAR2=“ World” VAR3=“VAR1VAR2” echo “VAR3”`. : *Hello,World* — : `VAR1 = “Hello,”VAR1+ = “World” echo“VAR1”`. Результат: *Hello, World*

3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?

Команда `seq` в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: `seq LAST`: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение `is` не выдает. `seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. `seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT.



Если LAST меньше, чем FIRST, он не производит вывод. `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT являются необязательными. `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными. `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.

4. Какой результат даст вычисление выражения  $\$((10/3))$ ?

Результатом данного выражения  $\$((10/3))$  будет 3, потому что это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

Отличия командной оболочки zsh от bash: В zsh более быстрое автодополнение для cd с помощью Tab В zsh существует калькулятор zcalc, способный выполнять вычисления внутри терминала В zsh поддерживаются числа с плавающей запятой В zsh поддерживаются структуры данных «хэш» В zsh поддерживается раскрытие полного пути на основе неполных данных В zsh поддерживается замена части пути В zsh есть возможность отображать разделенный экран, такой же как разделенный экран vim

6. Проверьте, верен ли синтаксис данной конструкции

```
for ((a=1; a <= LIMIT; a++))
```

Синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

Преимущества и недостатки скриптового языка bash:

- Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS
  - Удобное перенаправление ввода/вывода
  - Большое количество команд для работы с файловыми системами Linux
  - Можно писать собственные скрипты, упрощающие работу в Linux
- Недостатки скриптового языка bash:
- Дополнительные библиотеки других языков позволяют выполнить больше действий
  - Bash не является языком общего назначения - Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта
  - Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

## 5 Выводы

В ходе выполнения лабораторной работы мы изучили основы программирования в оболочке ОС UNIX а также научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## **6 Список литературы**

Не пользовалась сайтами.