

Лабораторная работа №12

Отчёт к лабораторной работе

Зайцева Анна Дмитриевна

Table of Contents

Цель работы

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

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.

Выполнение лабораторной работы

1. Я открыла `emacs` (команда: `emacs`) (Рис. [-@fig:001]):

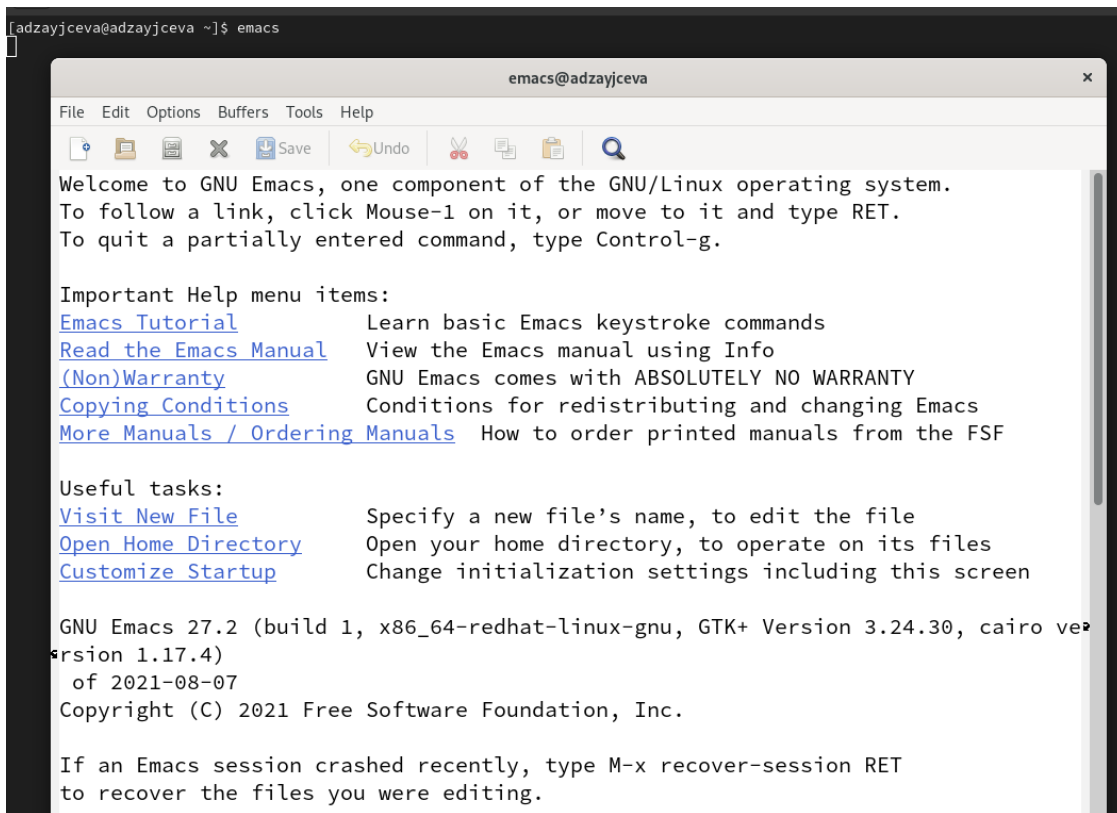


Рис. 1

Создала файл `sem.sh` с помощью комбинации `Ctrl-x Ctrl-f` (C-x C-f). Написала командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени `t1` дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени `t2 < t1`, 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.

sem.sh:

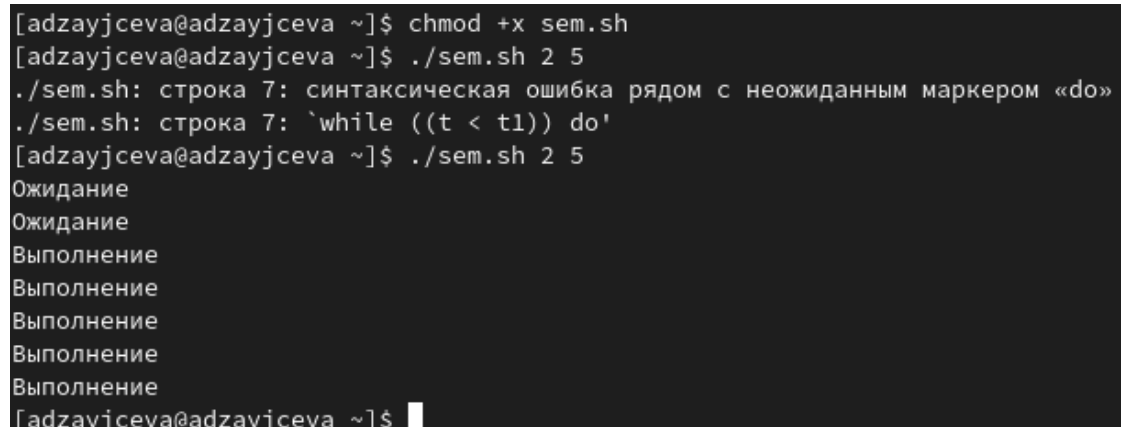
```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
```

```

s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))
while ((t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done

```

Добавила право на исполнение файла (команда: *chmod +x sem.sh*) и запустила его (команда: *./sem.sh 2 5*). Скрипт работает корректно (Рис. [-@fig:002]):



```

[adzayjceva@adzayjceva ~]$ chmod +x sem.sh
[adzayjceva@adzayjceva ~]$ ./sem.sh 2 5
./sem.sh: строка 7: синтаксическая ошибка рядом с неожиданным маркером «do»
./sem.sh: строка 7: `while ((t < t1)) do'
[adzayjceva@adzayjceva ~]$ ./sem.sh 2 5
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
[adzaviceva@adzaviceva ~]$

```

Рис. 2

Я доработала код программы, чтобы имелась возможность взаимодействия трёх и более процессов.

sem.sh (modified):

```

#!/bin/bash
function ojidaniye
{
s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done
}
function vipolneniye
{
s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))

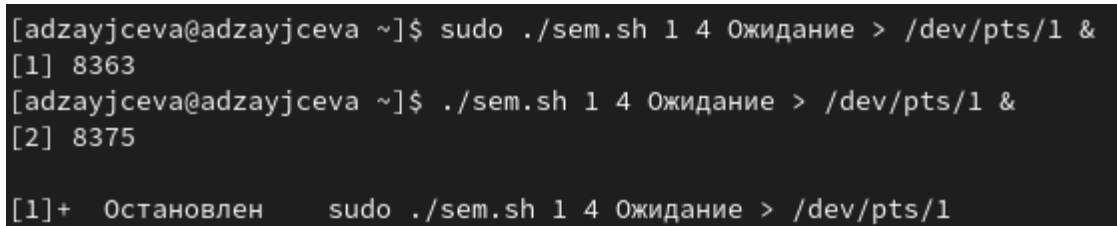
```

```

while ((t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo ""
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ojidaniye
    fi
    if [ "$command" == "Выполнение" ]
    then vipolneniye
    fi
    echo "Следующее действие: "
    read command
done

```

Запустила её (команда: *sudo ./sem.sh 1 4 Ожидание > /dev/pts/1 &*). Скрипт работает корректно (Рис. [-@fig:003]):



```

[adzayjceva@adzayjceva ~]$ sudo ./sem.sh 1 4 Ожидание > /dev/pts/1 &
[1] 8363
[adzayjceva@adzayjceva ~]$ ./sem.sh 1 4 Ожидание > /dev/pts/1 &
[2] 8375

[1]+  Остановлен      sudo ./sem.sh 1 4 Ожидание > /dev/pts/1

```

Рис. 3

2. Изучила 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 (Рис. [-@fig:004]):

```
[adzayjceva@adzayjceva ~]$ cd /usr
[adzayjceva@adzayjceva usr]$ cd /share/man/man1
bash: cd: /share/man/man1: Нет такого файла или каталога
[adzayjceva@adzayjceva usr]$ cd share/man/man1
[adzayjceva@adzayjceva man1]$ ls -a

. 1.gz gh-issue-edit.1.gz lptions.1.gz podman-system-migrate.1.gz
1.1.gz gh-issue-list.1.gz lpq.1.gz podman-system-prune.1.gz
1.1.gz gh-issue-reopen.1.gz lpq-cups.1.gz podman-system-renumber.1.gz
sh.1.gz gh-issue-status.1.gz lpr.1.gz podman-system-reset.1.gz
shrt.1.gz gh-issue-transfer.1.gz lprm-cups.1.gz podman-system-service.1.gz
shrt-action-analyze-backtrace.1.gz gh-label.1.gz lpstat.1.gz podman-tag.1.gz
shrt-action-analyze-c.1.gz gh-label-create.1.gz lpstat-cups.1.gz podman-top.1.gz
shrt-action-analyze-cpp-local.1.gz gh-label-list.1.gz ls.1.gz podman-umount.1.gz
shrt-action-analyze-core.1.gz ghostscript.1.gz lsattr.1.gz podman-umount.1.gz
shrt-action-analyze-java.1.gz gh-pr.1.gz lscpu.1.gz podman-unpause.1.gz
shrt-action-analyze-ops.1.gz gh-pr-checkout.1.gz lscpu.1.gz podman-unshare.1.gz
shrt-action-analyze-python.1.gz gh-pr-checks.1.gz lsinitrd.1.gz podman-untag.1.gz
shrt-action-analyze-vcare.1.gz gh-pr-class.1.gz lsipm.1.gz podman-version.1.gz
shrt-action-analyze-vulnerability.1.gz gh-pr-comment.1.gz lsirq.1.gz podman-volume.1.gz
shrt-action-check-ops-for-hermerror.1.gz gh-pr-create.1.gz lslogins.1.gz podman-volume-expart.1.gz
shrt-action-find-badhi-update.1.gz gh-pr-diff.1.gz lsmem.1.gz podman-volume-export.1.gz
shrt-action-generate-backtrace.1.gz gh-pr-edit.1.gz lsmod.1.gz podman-volume-import.1.gz
shrt-action-generate-core-backtrace.1.gz gh-pr-list.1.gz lsmt.1.gz podman-volume-inspect.1.gz
shrt-action-install-debuginfo.1.gz gh-pr-mega.1.gz lsua.1.gz podman-volume-ls.1.gz
shrt-action-list-dns.1.gz gh-pr-ready.1.gz lsuc.1.gz podman-volume-prune.1.gz
shrt-action-notify.1.gz gh-pr-reopen.1.gz lsusradd.1.gz podman-volume-rw.1.gz
shrt-action-perform-cpp-analysis.1.gz gh-pr-review.1.gz lsuserdel.1.gz podman-wait.1.gz
shrt-action-save-package-data.1.gz gh-pr-view.1.gz lsusermod.1.gz portablectl.1.gz
shrt-action-trim-files.1.gz gh-release.1.gz lsusermod.1.gz p057.1.gz
shrt-applet.1.gz gh-release-create.1.gz ls2to1.1.gz p062.1.gz
shrt-auto-reporting.1.gz gh-release-delete.1.gz ls2to1.1.gz p063.1.gz
shrt-badhi.1.gz gh-release-delete-asset.1.gz ls2to1.1.gz p064.1.gz
shrt-ctl.1.gz gh-release-download.1.gz ls2to1.1.gz p065.1.gz
shrt-dump-journal-core.1.gz gh-release-list.1.gz ls2to1.1.gz p066.1.gz
shrt-dump-journal-ops.1.gz gh-release-upload.1.gz ls2to1.1.gz p067.1.gz
shrt-dump-journal-sg.1.gz gh-release-view.1.gz ls2to1.1.gz p068.1.gz
shrt-dump-ops.1.gz gh-repo.1.gz ls2to1.1.gz p069.1.gz
shrt-dump-sg.1.gz gh-repo-archive.1.gz ls2to1.1.gz p070.1.gz
shrt-dump-sg.1.gz gh-repo-clone.1.gz ls2to1.1.gz p071.1.gz
```

Рис. 4

Создала файл `man.sh` с помощью комбинации `Ctrl-x Ctrl-f` (C-x C-f). Реализовала команду `man` с помощью командного файла. 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`.

man.sh:

```
#!/bin/bash
c=$1 # Инициализация названия команды
if [ -f /usr/share/man/man1/$c.1.gz ] # Проверка существования справки
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less # Распаковка архива со
    справкой, если она есть
else
    echo "Справки нет"
fi
```

Добавила право на исполнение файла (команда: `chmod +x man.sh`) и запустила скрипт несколько раз (команды: `./man.sh make` и `./man.sh sg`) (Рис. [-@fig:005]) Скрипт работает корректно (Рис. [-@fig:006])(Рис. [-@fig:007]):

```
[adzayjceva@adzayjceva ~]$ chmod +x man.sh
[adzayjceva@adzayjceva ~]$ ./man.sh make
[adzayjceva@adzayjceva ~]$ ./man.sh sg
[adzayjceva@adzayjceva ~]$
```

Рис. 5

```

.TH MAKE 1 "28 February 2016" "GNU" "User Commands"
.SH NAME
make \- GNU make utility to maintain groups of programs
.SH SYNOPSIS
.B make
[\fIOPTION\fr]... [\fITARGET\fr]...
.SH DESCRIPTION
.LP
The
.I make
utility will determine automatically which pieces of a large program need to
be recompiled, and issue the commands to recompile them. The manual describes
the GNU implementation of
.BR make ,
which was written by Richard Stallman and Roland McGrath, and is currently
maintained by Paul Smith. Our examples show C programs, since they are very
common, but you can use
.B make
with any programming language whose compiler can be run with a shell command.
In fact,
.B make
is not limited to programs. You can use it to describe any task where some
files must be updated automatically from others whenever the others change.
.LP
To prepare to use
.BR make ,
you must write a file called the
.I makefile
that describes the relationships among files in your program, and the states
the commands for updating each file. In a program, typically the executable
file is updated from object files, which are in turn made by compiling source
files.
.LP
Once a suitable makefile exists, each time you change some source files,
this simple shell command:
.sp 1
.RS
.B make
.RE
.sp 1
suffices to perform all necessary recompilations.
The
:

```

Puc. 6

```

.\" t
.\" Title: sg
.\" Author: Julianne Frances Haugh
.\" Generator: DocBook XSL Stylesheets vsnapshot <http://docbook.sf.net/>
.\" Date: 08/17/2021
.\" Manual: User Commands
.\" Source: shadow-utils 4.9
.\" Language: English
.\"
.TH "SG" "1" "08/17/2021" "shadow-utils 4.9" "User Commands"
.\" -----
.\" * Define some portability stuff
.\" -----
.\" ~~~~~
.\" http://bugs.debian.org/507673
.\" http://lists.gnu.org/archive/html/groff/2009-02/msg00013.html
.\" ~~~~~
.ie \n(.g .ds Aq \(aq
.el .ds Aq '
.\" -----
.\" * set default formatting
.\" -----
.\" disable hyphenation
.nh
.\" disable justification (adjust text to left margin only)
.ad l
.\" -----
.\" * MAIN CONTENT STARTS HERE *
.\" -----
.SH "NAME"
sg \- execute command as different group ID
.SH "SYNOPSIS"
.HP \w'\fBsg\fR\ 'u
\fBsg\fR [\-] [group\ [\-c\ ]\ command]
.SH "DESCRIPTION"
.PP
The
\fBsg\fR
command works similar to
\fBnewgrp\fR
but accepts a command\&. The command will be executed with the
/bin/sh
:

```

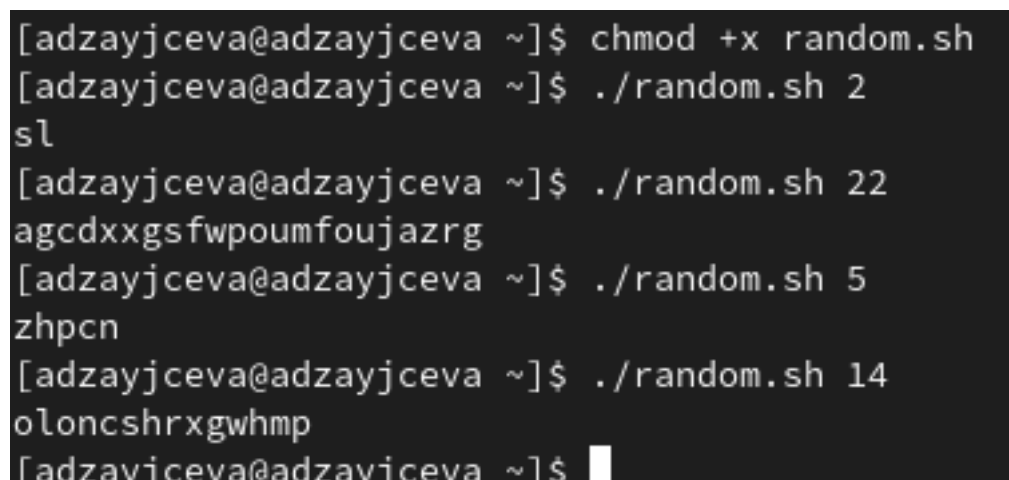
Рис. 7

3. Создала файл random.sh с помощью комбинации Ctrl-x Ctrl-f (C-x C-f). Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учла, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

random.sh:

```
#!/bin/bash
c=$1 # Инициализация количества символов
for (( i=0; i<$c; i++ )) #Цикл вывода нужного количества символов
do
    (( char=$RANDOM%26+1 )) # Случайные номера от 1 до 26
    case $char in # Вывод символа с помощью оператора выбора
        1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) 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
```

Добавила право на исполнение файла (команда: `chmod +x random.sh`) и запустила скрипт несколько раз (команды: `./random.sh 2`, `./random.sh 22`, `./random.sh 5` и `./random.sh 14`). Скрипт работает корректно (Рис. [-@fig:008]):



```
[adzayjceva@adzayjceva ~]$ chmod +x random.sh
[adzayjceva@adzayjceva ~]$ ./random.sh 2
sl
[adzayjceva@adzayjceva ~]$ ./random.sh 22
agcdxxgsfwpoumfoujazrg
[adzayjceva@adzayjceva ~]$ ./random.sh 5
zhpcn
[adzayjceva@adzayjceva ~]$ ./random.sh 14
oloncshrxgwhmp
[adzayjceva@adzayjceva ~]$
```

Рис. 8

Ответы на контрольные вопросы

1. `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 ["$1" != "exit"]`
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:
 - Первый: `VAR1="Hello,"VAR2=" World" VAR3="VAR1VAR2" echo "$VAR3"` Результат: Hello, World

- Второй: VAR1="Hello," VAR1+=" World" echo "\$VAR1" Результат: Hello, World
- 3. 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. Результатом данного выражения $\$(10/3)$ будет 3, потому что это целочисленное деление без остатка.
- 5. Отличия командной оболочки zsh от bash:
 - В zsh более быстрое автодополнение для cdc 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. for((a=1; a<= LIMIT; a++)) синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().
- 7. **Преимущества 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.

Вывод

В ходе лабораторной работы я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.