

Отчёт по лабораторной работе №12

Операционные системы

Обрезкова Анастасия Владимировна

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	7
4	Выводы	16
5	Контрольные вопросы	17

Список иллюстраций

3.1	Создала файл и написала первый скрипт	8
3.2	Проверила первый скрипт	8
3.3	Изменила первый скрипт	9
3.4	Изменила первый скрипт	10
3.5	Проверила первый скрипт повторно	10
3.6	Содержимое каталога /usr/share/man/man1	11
3.7	Создала файл и написала второй скрипт	12
3.8	Проверила второй скрипт	12
3.9	Проверила второй скрипт	13
3.10	Создал файл и написал третий скрипт	14
3.11	Проверил третий скрипт	15

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

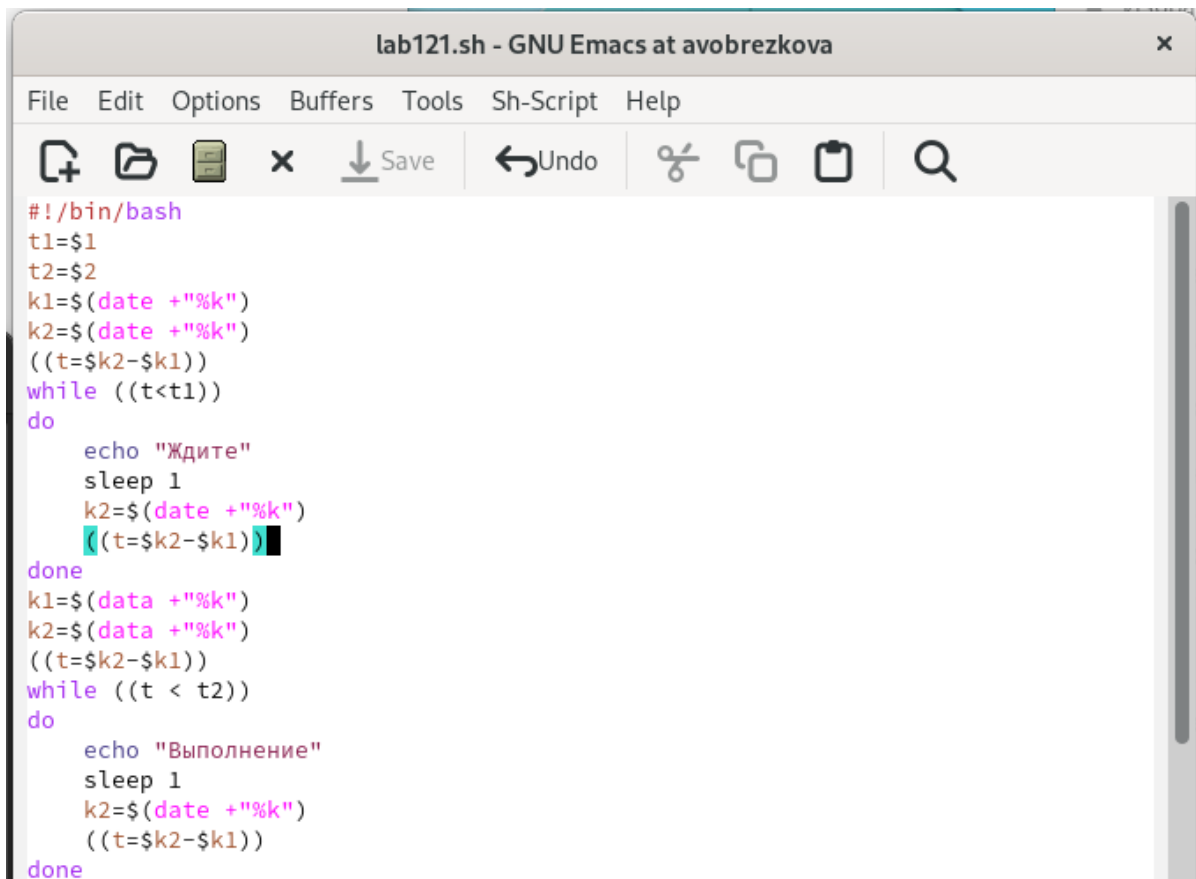
1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию 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

1. Написала 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).

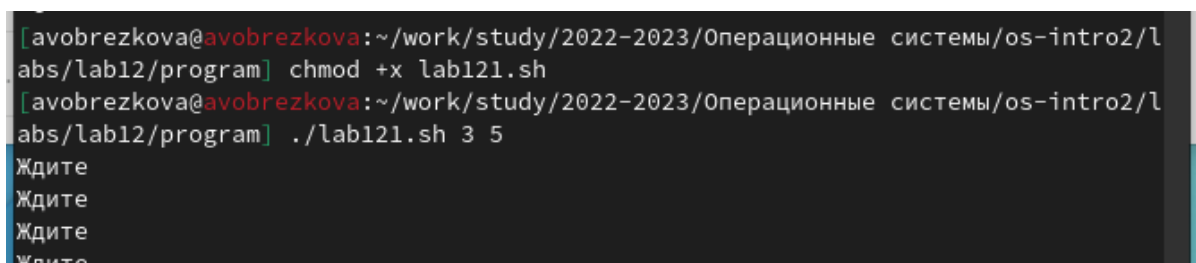
Для данной задачи я создала файл: lab121.sh и написала соответствующий скрипт (рис. -3.1).



```
#!/bin/bash
t1=$1
t2=$2
k1=$(date +%k)
k2=$(date +%k)
((t=$k2-$k1))
while ((t<t1))
do
    echo "Ждите"
    sleep 1
    k2=$(date +%k)
    ((t=$k2-$k1))
done
k1=$(date +%k)
k2=$(date +%k)
((t=$k2-$k1))
while ((t < t2))
do
    echo "Выполнение"
    sleep 1
    k2=$(date +%k)
    ((t=$k2-$k1))
done
```

Рис. 3.1: Создала файл и написала первый скрипт

Далее я проверила работу написанного скрипта (команда «./lab121.sh 3 5»), предварительно добавив право на исполнение файла (команда «chmod +x lab121.sh»). Скрипт работает корректно. (рис. -3.2).

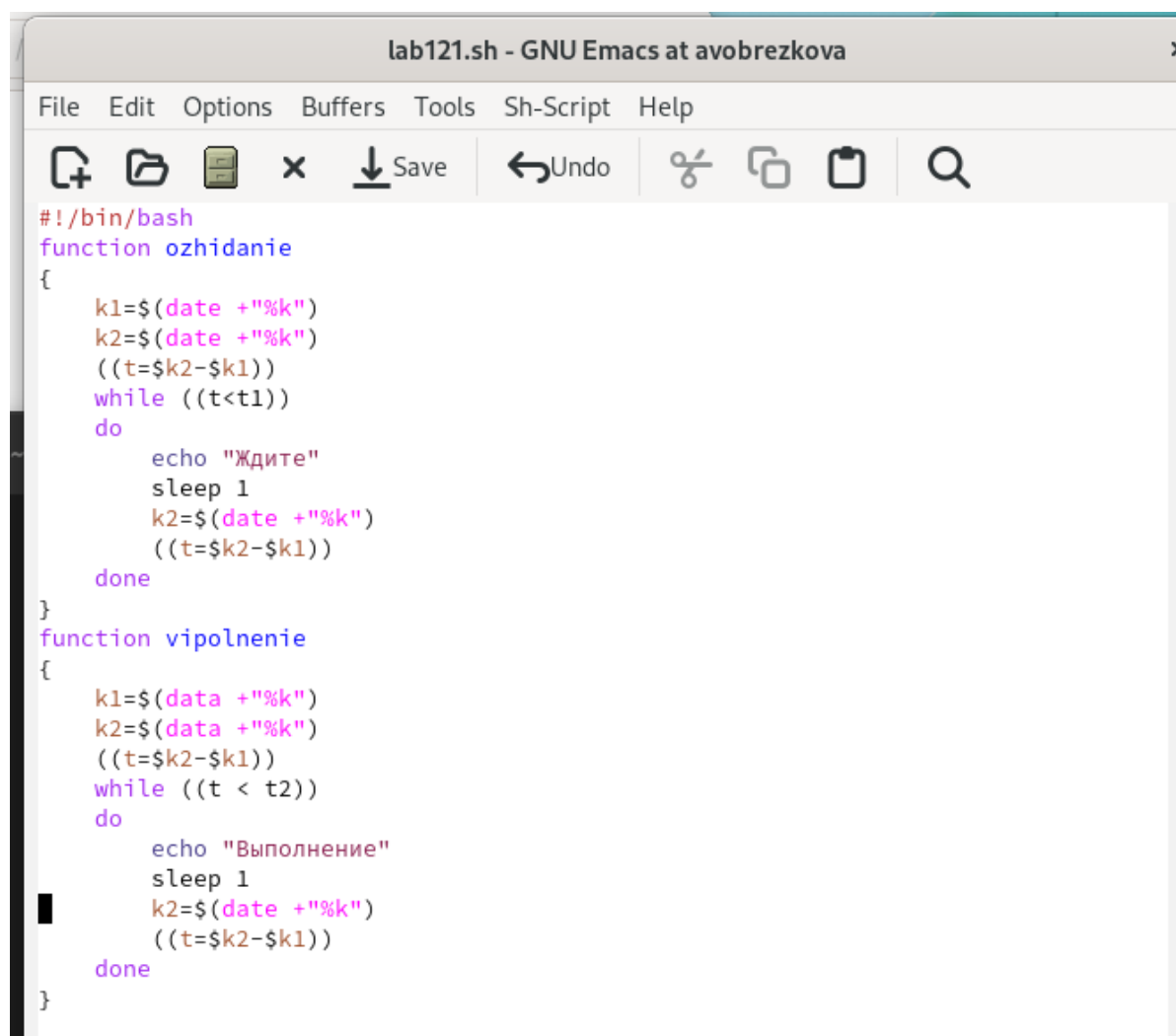


```
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program] chmod +x lab121.sh
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program] ./lab121.sh 3 5
Ждите
Ждите
Ждите
Ждите
```

Рис. 3.2: Проверила первый скрипт

После этого я изменила скрипт так, чтобы его можно было выполнять в несколь-

ких терминалах и проверила его работу. (рис. -3.3; рис. -3.4)



```
lab121.sh - GNU Emacs at avobrezkova
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
function ozhidanie
{
    k1=$(date +%k)
    k2=$(date +%k)
    ((t=k2-k1))
    while ((t<t1))
    do
        echo "Ждите"
        sleep 1
        k2=$(date +%k)
        ((t=k2-k1))
    done
}
function vipolnenie
{
    k1=$(date +%k)
    k2=$(date +%k)
    ((t=k2-k1))
    while ((t < t2))
    do
        echo "Выполнение"
        sleep 1
        k2=$(date +%k)
        ((t=k2-k1))
    done
}
```

Рис. 3.3: Изменила первый скрипт

```

}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ждите" ]
    then ozhidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done

```

Рис. 3.4: Изменила первый скрипт

Но ни одна команда не работала, так как мне было “Отказано в доступе”. При этом скрипт работает корректно. (рис. -3.5).

```

avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program$ ./lab121.sh Ждите > /dev/pts/1 &
[1] 5395
bash: /dev/pts/1: Отказано в доступе
[1]+  Выход 1      ./lab121.sh Ждите > /dev/pts/1
avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program$ ./lab121.sh 2 4 Ждите > /dev/pts/1 &
[1] 5403
avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program$ ./lab121.sh 3 4 Ждите > /dev/pts/2 &
[1] 5413
bash: /dev/pts/2: Отказано в доступе
[1]+  Выход 1      ./lab121.sh 3 4 Ждите > /dev/pts/2
avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program$

```

Рис. 3.5: Проверила первый скрипт повторно

2. Реализовала команду man с помощью командного файла. 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. 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. (рис. -3.6)

```
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program] cd /usr/share/man/man1
[avobrezkova@avobrezkova:/usr/share/man/man1] ls
.:1.gz
'[:1.gz'
a2ping.1.gz
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
mren.1.gz
msgattrib.1.gz
msgcat.1.gz
msgcmp.1.gz
msgcomm.1.gz
msgconv.1.gz
msgen.1.gz
msgexec.1.gz
msgfilter.1.gz
msgfmt.1.gz
msggrep.1.gz
msginit.1.gz
msgmerge.1.gz
msgunfmt.1.gz
msguniq.1.gz
mshortname.1.gz
mshowfat.1.gz
msxlint.1.gz
mthelp.1.gz
mtools.1.gz
mtoolstest.1.gz
mtrace.1.gz
mtx-babel.1.gz
mtx-base.1.gz
mtx-bibtex.1.gz
mtx-cache.1.gz
mtx-chars.1.gz
mtx-check.1.gz
```

Рис. 3.6: Содержимое каталога /usr/share/man/man1

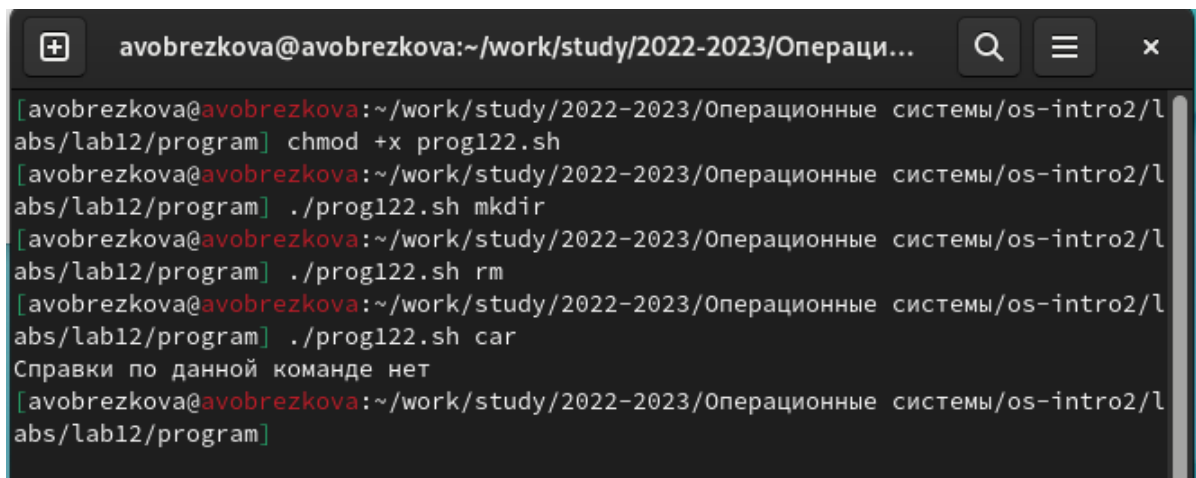
Для данной задачи я создала файл: prod122.sh и написала соответствующий скрипт (рис. -3.7).

fi" data-bbox="121 80 874 323"/>

```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.gz ]
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less
else
    echo "Справки по данной команде нет"
fi
```

Рис. 3.7: Создала файл и написала второй скрипт

Далее я проверила работу написанного скрипта, предварительно добавив право на исполнение файла. Скрипт работает корректно. (рис. -3.8; рис. -3.9)



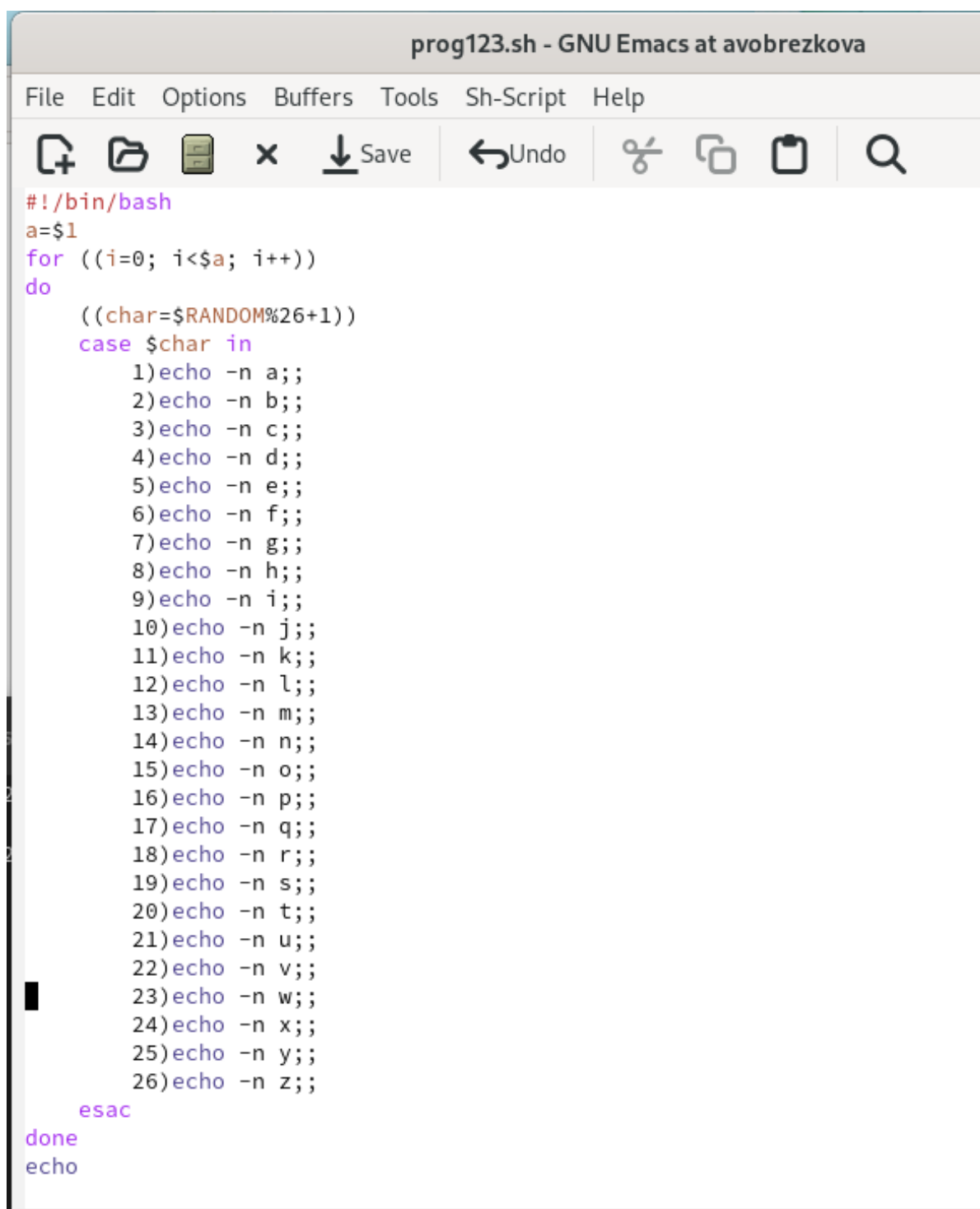
```
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program] chmod +x prog122.sh
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program] ./prog122.sh mkdir
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program] ./prog122.sh rm
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program] ./prog122.sh car
Справки по данной команде нет
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/lab12/program]
```

Рис. 3.8: Проверила второй скрипт

```
avobrezkova@avobrezkova:~/work/study/2022-2023/Операци...
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH MKDIR "1" "January 2023" "GNU coreutils 9.0" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\\fI\\,OPTION\\/\fR]... \\fI\\,DIRECTORY\\/\fR...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Create the DIRECTORY(ies), if they do not already exist.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\\fB\\-m\\fR, \\fB\\-\\-mode\\fR=\\fI\\,MODE\\/\fR
set file mode (as in chmod), not a=rwx \\- umask
.TP
\\fB\\-p\\fR, \\fB\\-\\-parents\\fR
no error if existing, make parent directories as needed,
with their file modes unaffected by any \\fB\\-m\\fR option.
.TP
\\fB\\-v\\fR, \\fB\\-\\-verbose\\fR
print a message for each created directory
:
```

Рис. 3.9: Проверила второй скрипт

- Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создала файл: prog123.sh и написал соответствующий скрипт. (рис. -3.10).



```
#!/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;;
        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
```

Рис. 3.10: Создал файл и написал третий скрипт

Далее я проверила работу написанного скрипта, предварительно добавив право

на исполнение файла. Скрипт работает корректно. (рис. -3.11).

```
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program] chmod +x prog123.sh
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program] ./prog123.sh 5
brnjl
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program] ./prog123.sh 9
qzfyzygxm
[avobrezkova@avobrezkova:~/work/study/2022-2023/Операционные системы/os-intro2/labs/lab12/program]
```

Рис. 3.11: Проверил третий скрипт

Данные изменения можно проверить по ссылке: https://github.com/avobrezkova/study_2022-2023_os-intro/tree/master/labs/lab12

4 Выводы

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

5 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 `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. `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