

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 13

дисциплина: Операционные системы

Студент: Хосе Фернадо Леон Атупанья

Группа: НПМбд-02-20

МОСКВА 2021 г.

Цель работы

Изучить основы программирования в оболочке операционной системы UNIX. Узнайте, как писать более сложные пакетные файлы, используя логические конструкции управления и циклы.

Задания

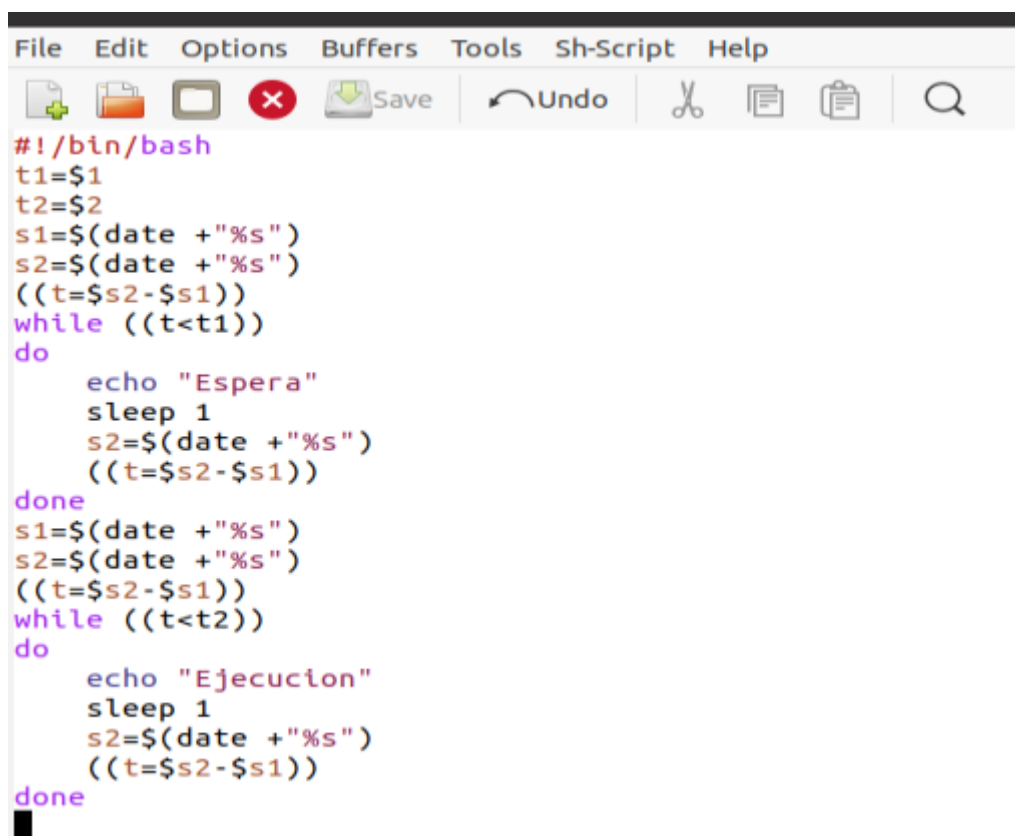
1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию 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. Написал 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). Для данной задачи я создал файл: semafor.sh и написал соответствующий скрипт (рис. -@fig:001).



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

{ #fig:001 width=70% }

Далее я проверил работу написанного скрипта (команда «./semafor.sh 3 5»), предварительно добавив право на исполнение файла (команда «chmod +x semafor.sh») (рис. -@fig:002). Скрипт работает корректно.

```
fernando@fernando-VirtualBox:~$ cd labor2
fernando@fernando-VirtualBox:~/labor2$ cd 2020-2021
fernando@fernando-VirtualBox:~/labor2/2020-2021$ cd OS
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS$ mkdir laboratory13
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS$ cd laboratory13
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ touch semafor.sh
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ emacs &
[1] 13434
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ chmod +x semafor.sh
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ ./semafor.sh
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ ./semafor.sh 3 5
Espera
Espera
Espera
Ejecucion
Ejecucion
Ejecucion
Ejecucion
Ejecucion
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$
```

{ #fig:001 width=70% }

После этого я изменил скрипт так, чтобы его можно было выполнять в нескольких терминалах (рис. -@fig:003) и проверил его работу (например, команда «./semafor.sh 2 4 Ожидание > /dev/pts/1 &»).

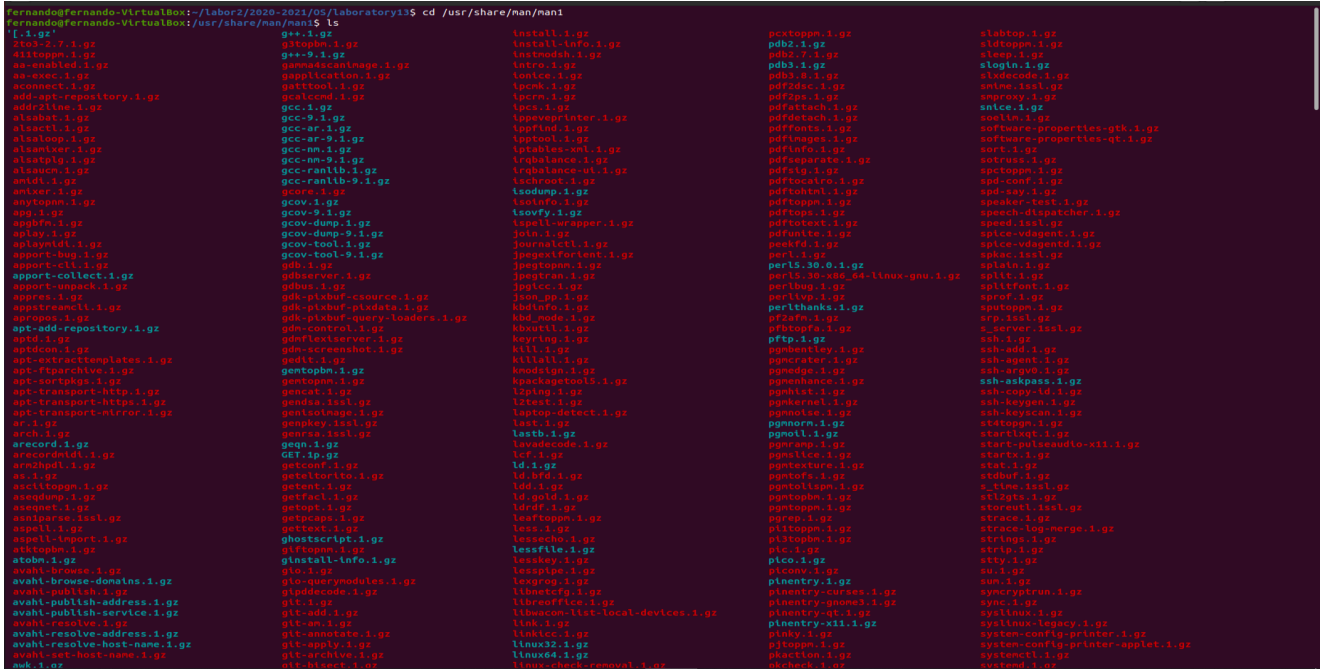
![Изменяем semafor.sh] semafor.sh](imag13/13.2.png){ #fig:001 width=70% }

К сожалению ни одна команда не работала, так как мне было "Отказано в доступе" (рис. -@fig:004). При этом скрипт работает корректно (команда «./semafor.sh 2 4 Ожидание»).

```
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ ./semafor.sh 2 4 Espera > &
bash: syntax error near unexpected token `&'
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ sudo ./semafor.sh 2 4 Espera > /dev/pts1 &
[2] 13516
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ bash: /dev/pts1: Permission denied
[2]+ Exit 1 sudo ./semafor.sh 2 4 Espera > /dev/pts1
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ ./semafor.sg 3 4 Espera > /dev/pts/2 &
[2] 13517
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ bash: /dev/pts/2: Permission denied
[2]+ Exit 1 ./semafor.sg 3 4 Espera > /dev/pts/2
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ ./semafor.sg 3 5 Salida > /dev/pts/2 &
[2] 13520
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ bash: /dev/pts/2: Permission denied
[2]+ Exit 1 ./semafor.sg 3 5 Salida > /dev/pts/2
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ ./semafor.sg 2 6 Ejecucion > /dev/pts/3 &
[2] 13521
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$ bash: /dev/pts/3: Permission denied
[2]+ Exit 1 ./semafor.sg 2 6 Ejecucion > /dev/pts/3
fernando@fernando-VirtualBox:~/labor2/2020-2021/OS/laboratory13$
```

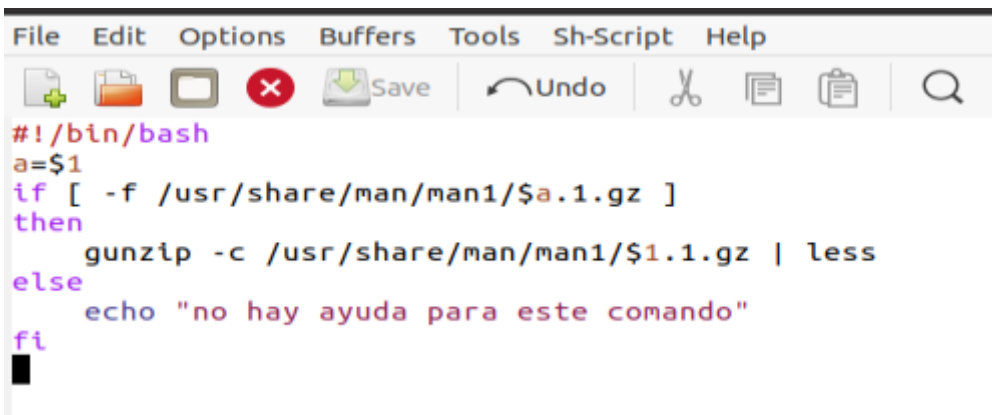
{ #fig:004 width=70% }

2. Реализовал команду man с помощью командного файла. Изучил содержимое каталога /usr/share/man/man1 (рис. -@fig:005). 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.



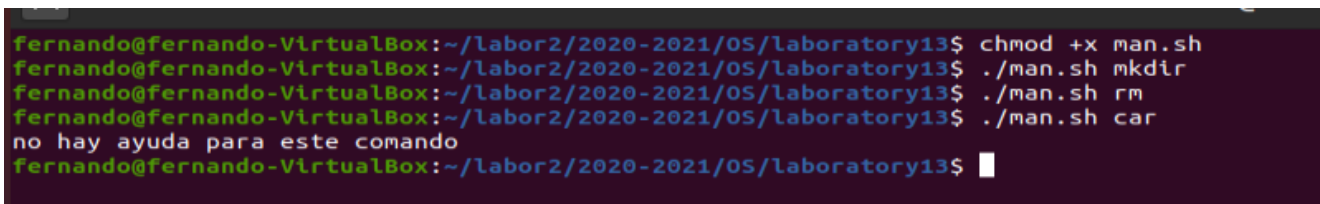
{ #fig:005 width=70% }

Для данной задачи я создал файл: man.sh и написал соответствующий скрипт (рис. -@fig:006).



{ #fig:006 width=70% }

Далее я проверил работу написанного скрипта (команды «./man.sh mkdir», «./man.sh rm» и «./man.sh car»), предварительно добавив право на исполнение файла (команда «chmod +x man.sh») (рис. -@fig:007) (рис. -@fig:008) (рис. -@fig:009). Скрипт работает корректно.



{ #fig:007 width=70% }

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "September 2019" "GNU coreutils 8.30" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fIOPTION\fR]\fR... \fIDIRECTORY\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=\fIMODE\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
.TP
\fb\-v\fr, \fb\--verbose\fr
print a message for each created directory
.TP
\fb\-Z\fr
set SELinux security context of each created directory
to the default type
.TP
\fb\--context\fr[=\fICTX\fR]
like \fb\-Z\fr, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
\fb\--help\fr
display this help and exit
.TP
\fb\--version\fr
output version information and exit
.SH AUTHOR
Written by David MacKenzie.
.SH "REPORTING BUGS"
GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
.br
Report mkdir translation bugs to <https://translationproject.org/team/>
.SH COPYRIGHT
Copyright \((c) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
.br
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
.SH "SEE ALSO"
mkdir(2)
.PP
.br
Full documentation at: <https://www.gnu.org/software/coreutils/mkdir>
.br
or available locally via: info \((aqcoreutils) mkdir invocation\((aq
(EN)

```

{ #fig:008 width=70% }

```

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH RM "1" "September 2019" "GNU coreutils 8.30" "User Commands"
.SH NAME
rm \- remove files or directories
.SH SYNOPSIS
.B rm
[\fI\,OPTION\|\fR]... [\fI\,FILE\|\fR]...
.SH DESCRIPTION
This manual page
documents the GNU version of
.BR rm .
.B rm
removes each specified file. By default, it does not remove
directories.
.P
If the \fI\-I\fR or \fI\--interactive=once\fR option is given,
and there are more than three files or the \fI\-r\fR, \fI\-R\fR,
or \fI\--recursive\fR are given, then
.B rm
prompts the user for whether to proceed with the entire operation. If
the response is not affirmative, the entire command is aborted.
.P
Otherwise, if a file is unwritable, standard input is a terminal, and
the \fI\-f\fR or \fI\--force\fR option is not given, or the
\fI\-i\fR or \fI\--interactive=always\fR option is given,
.B rm
prompts the user for whether to remove the file. If the response is
not affirmative, the file is skipped.
.SH OPTIONS
.PP
Remove (unlink) the FILE(s).
.TP
\fB\--f\fR, \fB\--force\fR
ignore nonexistent files and arguments, never prompt
.TP
\fB\--i\fR
prompt before every removal
.TP
\fB\--I\fR
prompt once before removing more than three files, or
when removing recursively; less intrusive than \fB\--i\fR,
while still giving protection against most mistakes
.TP
\fB\--interactive\fR[=\fI\,WHEN\|\fR]
prompt according to WHEN: never, once (\fB\--I\fR), or
always (\fB\--i\fR); without WHEN, prompt always
.TP
\fB\--one\--file\--system\fR
when removing a hierarchy recursively, skip any
directory that is on a file system different from
that of the corresponding command line argument
.TP
\fB\--no\--preserve\--root\fR
do not treat '/' specially
.TP
\fB\--preserve\--root\fR[=\fI\,all\|\fR]
do not remove '/' (default);
with 'all', reject any command line argument
on a separate device from its parent
.TP
\fB\--r\fR, \fB\--R\fR, \fB\--recursive\fR
:

```

{ #fig:009 width=70% }

- Используя встроенную переменную \$RANDOM, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создал файл: random.sh и написал соответствующий скрипт (рис. -@fig:010).

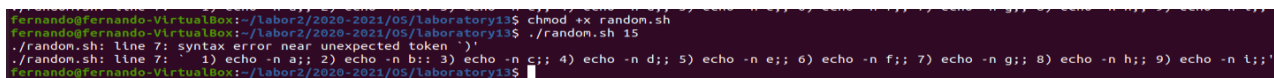
```

File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons] [Icons]
#!/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
done

```

{ #fig:010 width=70% }

Далее я проверил работу написанного скрипта (команда «./random.sh 15»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh») (рис. -@fig:011). Скрипт работает корректно.



{ #fig:011 width=70% }

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

Выводы

Во время этой лабораторной работы я изучил основы программирования в оболочке операционной системы UNIX, а также научился писать более сложные пакетные файлы с использованием логических конструкций управления и циклов.