

Отчёт лабораторной работы №13

Дисциплина: Операционные системы

Касьянов Даниил Владимирович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	15
4	Выводы	19
5	Библиография	20

Список таблиц

Список иллюстраций

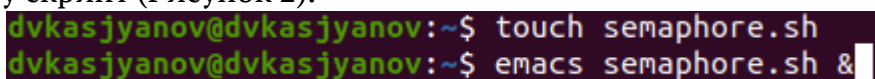
1 Цель работы

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

2 Выполнение лабораторной работы

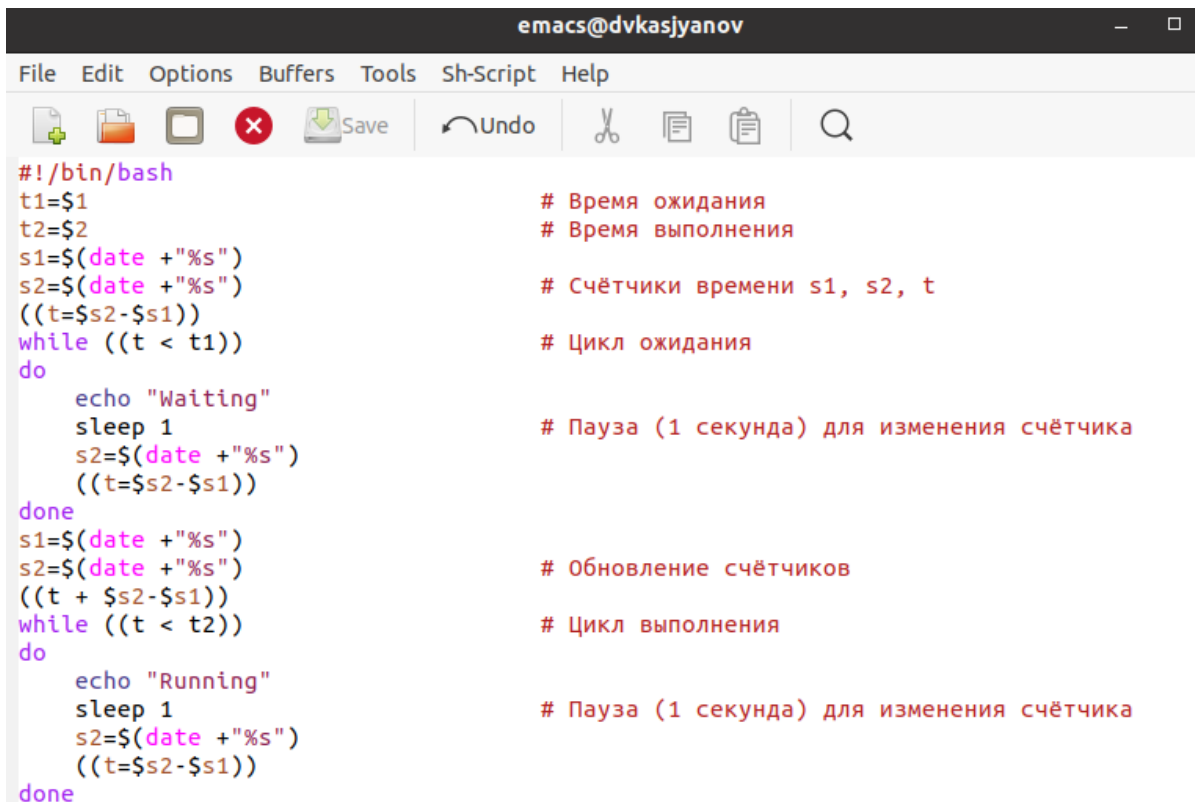
1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени **t1** дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени **t2<>t1**, также выдавая информацию 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.

Создаю файл **semaphore.sh** и открываю его, используя **emacs** (Рисунок 1). Пишу скрипт (Рисунок 2).



```
dvkasjyanov@dvkasjyanov:~$ touch semaphore.sh
dvkasjyanov@dvkasjyanov:~$ emacs semaphore.sh &
```

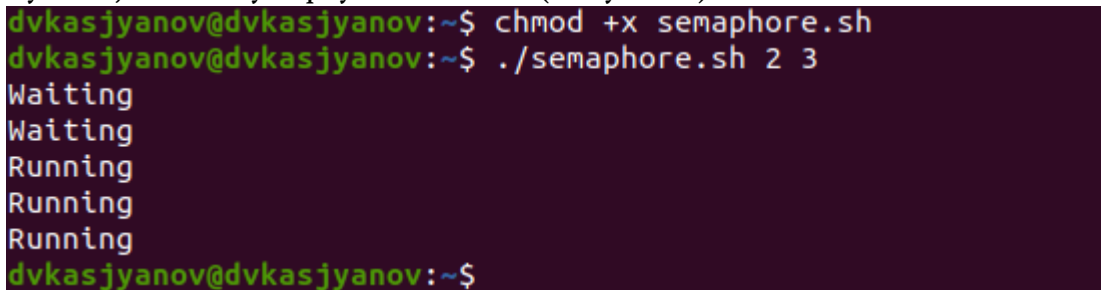
(Рисунок 1)



```
#!/bin/bash
t1=$1                                # Время ожидания
t2=$2                                # Время выполнения
s1=$(date +%s)                       # Счётчики времени s1, s2, t
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))                      # Цикл ожидания
do
    echo "Waiting"
    sleep 1                          # Пауза (1 секунда) для изменения счётчика
    s2=$(date +%s)
    ((t=$s2-$s1))
done
s1=$(date +%s)
s2=$(date +%s)                       # Обновление счётчиков
((t = $s2-$s1))
while ((t < t2))                      # Цикл выполнения
do
    echo "Running"
    sleep 1                          # Пауза (1 секунда) для изменения счётчика
    s2=$(date +%s)
    ((t=$s2-$s1))
done
```

(Рисунок 2)

Проверяю командный файл. Для этого добавляю ему право на выполнение и запускаю, используя аргументы 2 и 3 (Рисунок 3):



```
dvkasjyanov@dvkasjyanov:~$ chmod +x semaphore.sh
dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 2 3
Waiting
Waiting
Running
Running
Running
dvkasjyanov@dvkasjyanov:~$
```

(Рисунок 3)

Запускаю командный файл в терминале **tty4** в привилегированном режиме и в терминале **tty3** в фоновом режиме. Перенаправляю вывод из терминала **tty3** в **tty4**: `> /dev/tty4` (Рис. 4, 5).

```

dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 4 5 &
[1] 2825
dvkasjyanov@dvkasjyanov:~$ Waiting
Waiting
Waiting
Waiting
Running
Running
Running
Running
Running

[1]+  Завершён      ./semaphore.sh 4 5
dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 4 5 & > /dev/tty4
[1] 2873
dvkasjyanov@dvkasjyanov:~$ Waiting
Waiting
Waiting
Waiting
Running
Running
Running
Running
Running

```

(Рисунок 4)

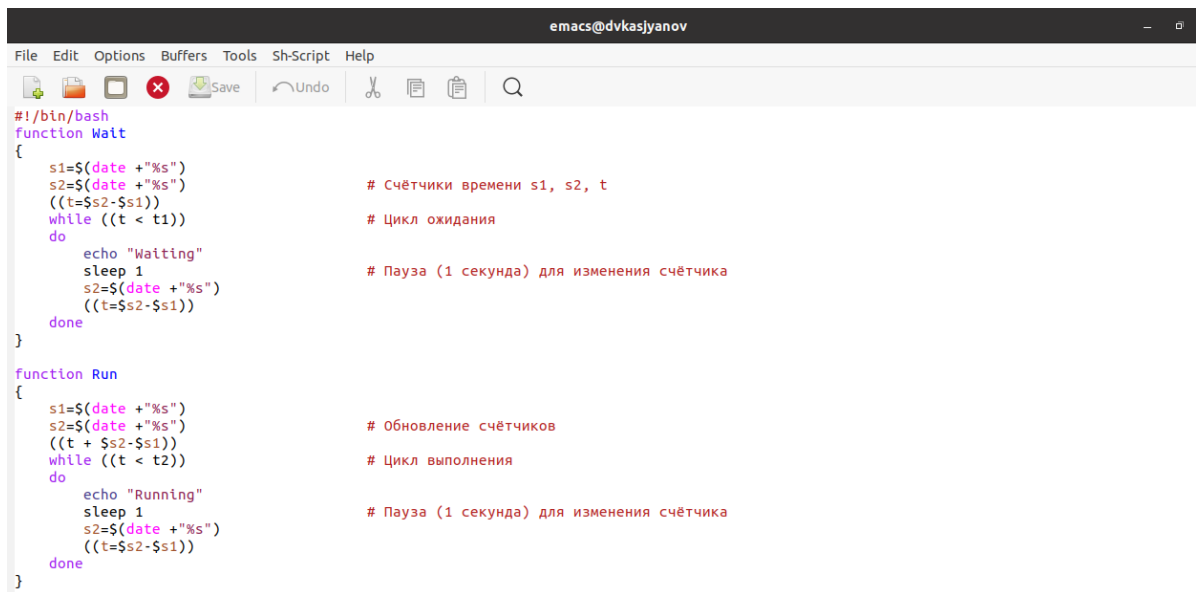
```

dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 5 6 $
Waiting
Waiting
Waiting
Waiting
Waiting
Running
Running
Running
Running
Running
Running
dvkasjyanov@dvkasjyanov:~$

```

(Рисунок 5)

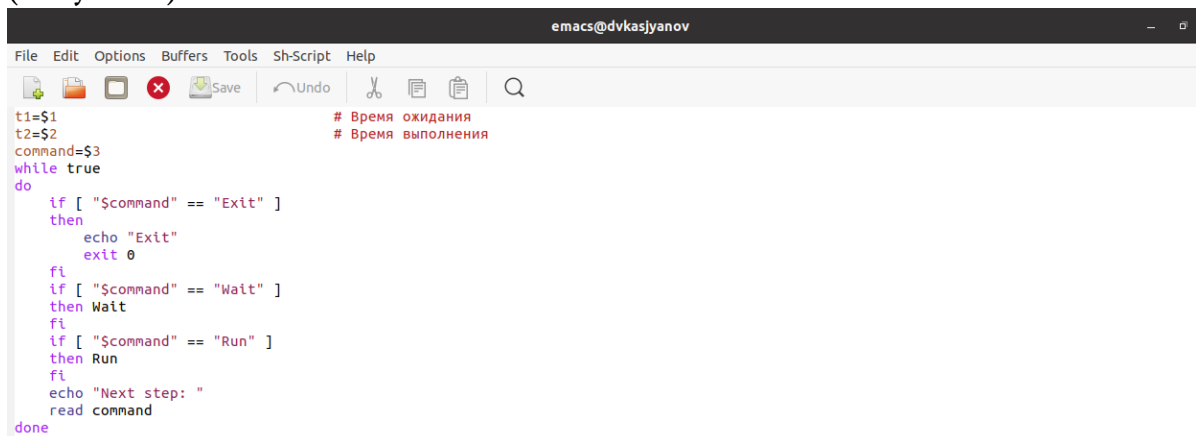
Доработаю программу для взаимодействия трёх и более процессов (Рис. 6, 7).



```
emacs@dvkasyanov
File Edit Options Buffers Tools Sh-Script Help
Save Undo Cut Copy Paste Find
#!/bin/bash
function Wait
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t < t1))
    do
        echo "Waiting"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}

function Run
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t = s2-s1))
    while ((t < t2))
    do
        echo "Running"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
```

(Рисунок 6)



```
emacs@dvkasyanov
File Edit Options Buffers Tools Sh-Script Help
Save Undo Cut Copy Paste Find
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Exit" ]
    then
        echo "Exit"
        exit 0
    fi
    if [ "$command" == "Wait" ]
    then
        Wait
    fi
    if [ "$command" == "Run" ]
    then
        Run
    fi
    echo "Next step: "
    read command
done
```

(Рисунок 7)

Проверю работу скрипта (Рис. 8-11).

```

dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 2 3 Wait & > /dev/tty3
[1] 3745
dvkasjyanov@dvkasjyanov:~$ Waiting
Waiting
Next step:
./semaphore.sh 2 3 Exit & > /dev/tty3
[2] 3752

[1]+  Остановлен    ./semaphore.sh 2 3 Wait
dvkasjyanov@dvkasjyanov:~$ Exit

[2]-  Завершён      ./semaphore.sh 2 3 Exit
dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 2 1 Run $ > /dev/tty4
^Z
[2]+  Остановлен    ./semaphore.sh 2 1 Run $ > /dev/tty4
dvkasjyanov@dvkasjyanov:~$
dvkasjyanov@dvkasjyanov:~$ ./semaphore.sh 3 1 Run & > /dev/tty5
[3] 3770
dvkasjyanov@dvkasjyanov:~$ Running
Next step:
./semaphore.sh 2 2 Exit & > /dev/tty5
[4] 3775

[3]+  Остановлен    ./semaphore.sh 3 1 Run
dvkasjyanov@dvkasjyanov:~$ Exit

[4]  Завершён      ./semaphore.sh 2 2 Exit
dvkasjyanov@dvkasjyanov:~$ _

```

(Рисунок 8)

```

Ubuntu 20.04.2 LTS dvkasjyanov tty3
dvkasjyanov login: dvkasjyanov
Password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-55-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

20 updates can be applied immediately.
Чтобы просмотреть дополнительные обновления выполните: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Sat Jun  5 00:08:55 MSK 2021 on tty2
dvkasjyanov@dvkasjyanov:~$

```

(Рисунок 9)

```

Ubuntu 20.04.2 LTS dvkasjyanov tty4

dvkasjyanov login: dvkasjyanov
Password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-55-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

20 updates can be applied immediately.
Чтобы просмотреть дополнительные обновления выполните: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Sat Jun  5 00:07:12 MSK 2021 on tty3
dvkasjyanov@dvkasjyanov:~$ Running
Next step:
./semaphore.sh 1 1 Wait
Waiting
Next step:
./semaphore.sh 1 2 Exit

```

(Рисунок 10)

```

Ubuntu 20.04.2 LTS dvkasjyanov tty5

dvkasjyanov login: dvkasjyanov
Password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-55-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

20 updates can be applied immediately.
Чтобы просмотреть дополнительные обновления выполните: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Sat Jun  5 00:07:21 MSK 2021 on tty4
dvkasjyanov@dvkasjyanov:~$

```

(Рисунок 11)

Программа работает корректно.

2. Реализовать команду **man** с помощью командного файла. Изучить 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**.

Просматриваю содержимое каталога **/usr/share/man/man1** (Рисунок 12).

```

dvkasjyanov@dvkasjyanov:~$ ls /usr/share/man/man1
'[.1.gz'
aa-enabled.1.gz
aa-exec.1.gz
aconnect.1.gz
add-apt-repository.1.gz
addr2line.1.gz
alsabat.1.gz
alsactl.1.gz
alsaloop.1.gz
alsamixer.1.gz
alsatplg.1.gz
alsaucm.1.gz
amidi.1.gz
amixer.1.gz
apg.1.gz
apgbfm.1.gz
aplay.1.gz
aplaymidi.1.gz
apport-bug.1.gz
apport-cli.1.gz
apport-collect.1.gz
apport-unpack.1.gz
appres.1.gz
appstreamcli.1.gz
apropos.1.gz
apt-add-repository.1.gz
aptd.1.gz
aptdcon.1.gz
apt-extracttemplates.1.gz
apt-ftpparchive.1.gz
apt-sortpkgs.1.gz
apt-transport-http.1.gz
apt-transport-https.1.gz
apt-transport-mirror.1.gz
ar.1.gz
arch.1.gz
arecord.1.gz
git-sh-setup.1.gz
git-sparse-checkout.1.gz
git-stage.1.gz
git-stash.1.gz
git-status.1.gz
git-strip-space.1.gz
git-submodule.1.gz
git-subtree.1.gz
git-switch.1.gz
git-symbolic-ref.1.gz
git-tag.1.gz
git-unpack-file.1.gz
git-unpack-objects.1.gz
git-update-index.1.gz
git-update-ref.1.gz
git-update-server-info.1.gz
git-upload-archive.1.gz
git-upload-pack.1.gz
git-var.1.gz
git-verify-commit.1.gz
git-verify-pack.1.gz
git-verify-tag.1.gz
gitweb.1.gz
git-web--browse.1.gz
git-whatchanged.1.gz
git-worktree.1.gz
git-write-tree.1.gz
glib-compile-schemas.1.gz
gnome-calculator.1.gz
gnome-control-center.1.gz
gnome-disk-image-mounter.1.gz
gnome-disks.1.gz
gnome-extensions.1.gz
gnome-help.1.gz
gnome-keyring.1.gz
gnome-keyring-3.1.gz
gnome-keyring-daemon.1.gz
pinentry-curses.1.gz
pinentry-gnome3.1.gz
pinentry-x11.1.gz
pinky.1.gz
pkaction.1.gz
pkcheck.1.gz
pkcon.1.gz
pkcs12.1.gz
pkcs7.1.gz
pkcs8.1.gz
pkexec.1.gz
pkey.1.gz
pkeyparam.1.gz
pkeyutil.1.gz
pkg-config.1.gz
pkill.1.gz
pkmon.1.gz
pkttyagent.1.gz
pl2pm.1.gz
pldd.1.gz
plog.1.gz
plymouth.1.gz
pmap.1.gz
pnm2ppa.1.gz
pod2html.1.gz
pod2man.1.gz
pod2text.1.gz
pod2usage.1.gz
podchecker.1.gz
podselect.1.gz
poff.1.gz
pon.1.gz
POST.1p.gz
ppdc.1.gz
ppdhtml.1.gz
ppdl.1.gz
ppdmerge.1.gz

```

(Рисунок 12)

Создаю файл **man.sh** и открываю его, используя **emacs** (Рисунок 13). Пишу скрипт (Рисунок 14).

```

dvkasjyanov@dvkasjyanov:~$ touch man.sh
dvkasjyanov@dvkasjyanov:~$ emacs man.sh

```

(Рисунок 13)

```

emacs@dvkasjyanov
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons] Search
#!/bin/bash
command=$1
if [ -f /usr/share/man/man1/$command.1.gz ]
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less
else
    echo "There is no manual for this command"
fi
# Команда
# Проверка наличия архива со справкой по команде
# Распаковка архива и вывод справки
# Сообщение об отсутствии мануала

```

(Рисунок 14)

Проверю работу скрипта.

Выведу справку по команде **rmdir** (Рис. 15, 16).

```

dvkasjyanov@dvkasjyanov:~$ chmod +x man.sh
dvkasjyanov@dvkasjyanov:~$ ./man.sh rmdir

```

(Рисунок 15)

```

.\\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH RMDIR "1" "September 2019" "GNU coreutils 8.30" "User Commands"
.SH NAME
rmmdir \- remove empty directories
.SH SYNOPSIS
.B rmmdir
[\fI\,OPTION\|\fR]... \fI\,DIRECTORY\|\fR...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Remove the DIRECTORY(ies), if they are empty.
.HP
\fB\-\-ignore\-\fail\-\on\-\non\-\empty\|fR
.IP
ignore each failure that is solely because a directory
.IP
is non\-\empty
.TP
\fB\-\p\|fR, \fB\-\-parents\|fR
remove DIRECTORY and its ancestors; e.g., 'rmmdir \fB\-\p\|fR a/b/c' is
similar to 'rmmdir a/b/c a/b a'
.TP
\fB\-\v\|fR, \fB\-\-verbose\|fR
output a diagnostic for every directory processed
.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 rmmdir translation bugs to <https://translationproject.org/team/>
.SH COPYRIGHT
:

```

(Рисунок 16)

Попробую использовать аргумент, который не является командой и, следовательно, не имеет мануала: `qwerty`. В результате получим сообщение об отсутствии мануала (Рисунок 17).

```

dvkasjyanov@dvkasjyanov:~$ ./man.sh qwerty
There is no manual for this command
dvkasjyanov@dvkasjyanov:~$

```

(Рисунок 17)

Программа работает корректно.

- Используя встроенную переменную **\$RANDOM**, написать командный файл, генерирующий случайную последовательность букв латинского алфавита. Учсть, что **\$RANDOM** выдаёт псевдослучайные числа в диапазоне от **0** до **32767**.

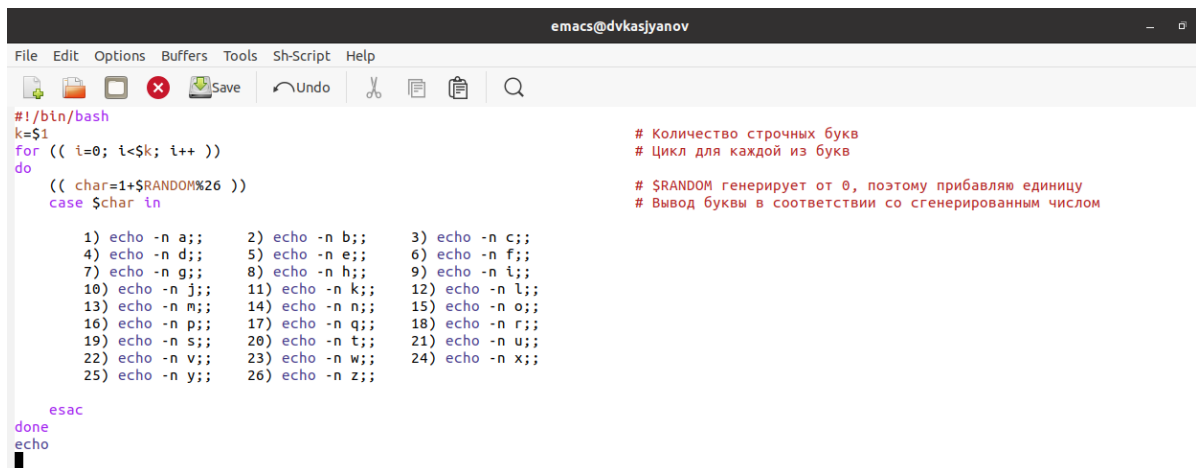
Создаю файл **random.sh** и открываю его, используя **emacs** (Рисунок 18). Пишу скрипт (Рисунок 19).

```

dvkasjyanov@dvkasjyanov:~$ touch random.sh
dvkasjyanov@dvkasjyanov:~$ emacs random.sh

```

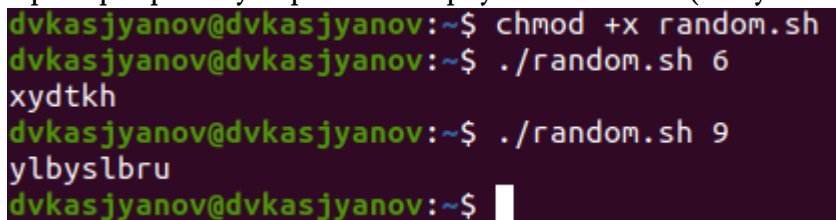
(Рисунок 18)



```
emacs@dvkasjyanov
File Edit Options Buffers Tools Sh-Script Help
Save Undo Cut Copy Paste Find
#!/bin/bash
k=$1
for (( i=0; i<$k; i++ ))
do
    (( char=1+$RANDOM%26 ))
    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
```

(Рисунок 19)

Проверю работу скрипта на аргументах 6 и 9 (Рисунок 20).



```
dvkasjyanov@dvkasjyanov:~$ chmod +x random.sh
dvkasjyanov@dvkasjyanov:~$ ./random.sh 6
xydtkh
dvkasjyanov@dvkasjyanov:~$ ./random.sh 9
ylbyslbru
dvkasjyanov@dvkasjyanov:~$
```

(Рисунок 20)

Программа работает корректно.

3 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) Для проведения конкатенации можно воспользоваться несколькими способами:

- `VAR1="Hello,"`
`VAR2=" World"`
`VAR3="$VAR1$VAR2"`
`echo "$VAR3"`
- `VAR1="Hello, "`
`VAR1+=" World"`
`echo "$VAR1"`

В обоих случаях результатом будет строка `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 верный, потому что при использовании двойных круглых скобок можно не писать **\$** перед переменными.

7) Преимущества скриптового языка **bash**:

- **Bash** является одним из самых распространенных скриптовых языков 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.
- Дополнительные библиотеки других языков позволяют выполнить больше действий.

Недостатки скриптового языка **bash**:

- **Bash** не является языком общего назначения.
- Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения.

- Скрипты, написанные на **bash**, нельзя запустить на других операционных системах без дополнительных действий.

4 Выводы

Я изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Библиография

Лабораторная работа №13 - “Программирование в командном процессоре ОС UNIX. Расширенное программирование”

Bash-скрипты: начало

НАПИСАНИЕ СКРИПТОВ НА BASH

“Advanced BashScripting Guide - Искусство программирования на языке сценариев командной оболочки” - Mendel Cooper