

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

### ОТЧЕТ

### ПО ЛАБОРАТОРНОЙ РАБОТЕ №13

дисциплина: *Операционные системы*

Студент: Мохаммад Амин Шоаибуллах Группа: НПИбд-02-20

МОСКВА

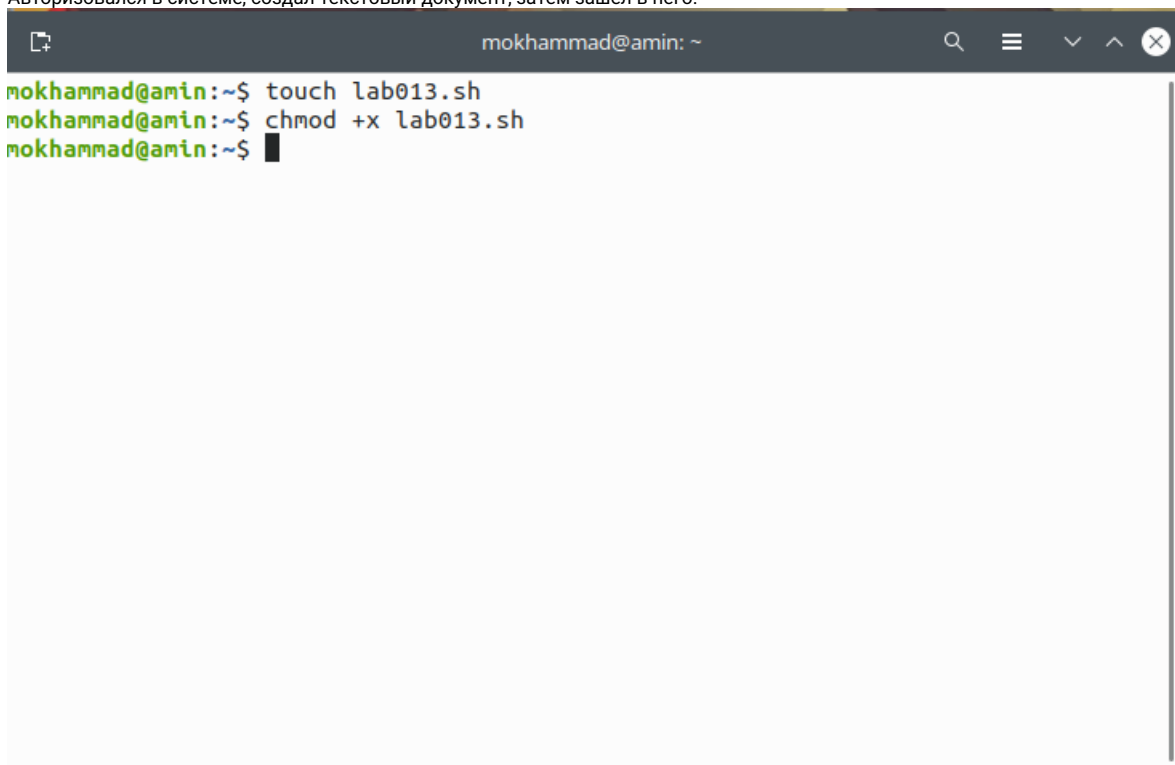
2021 г

### *Цель работы:*

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

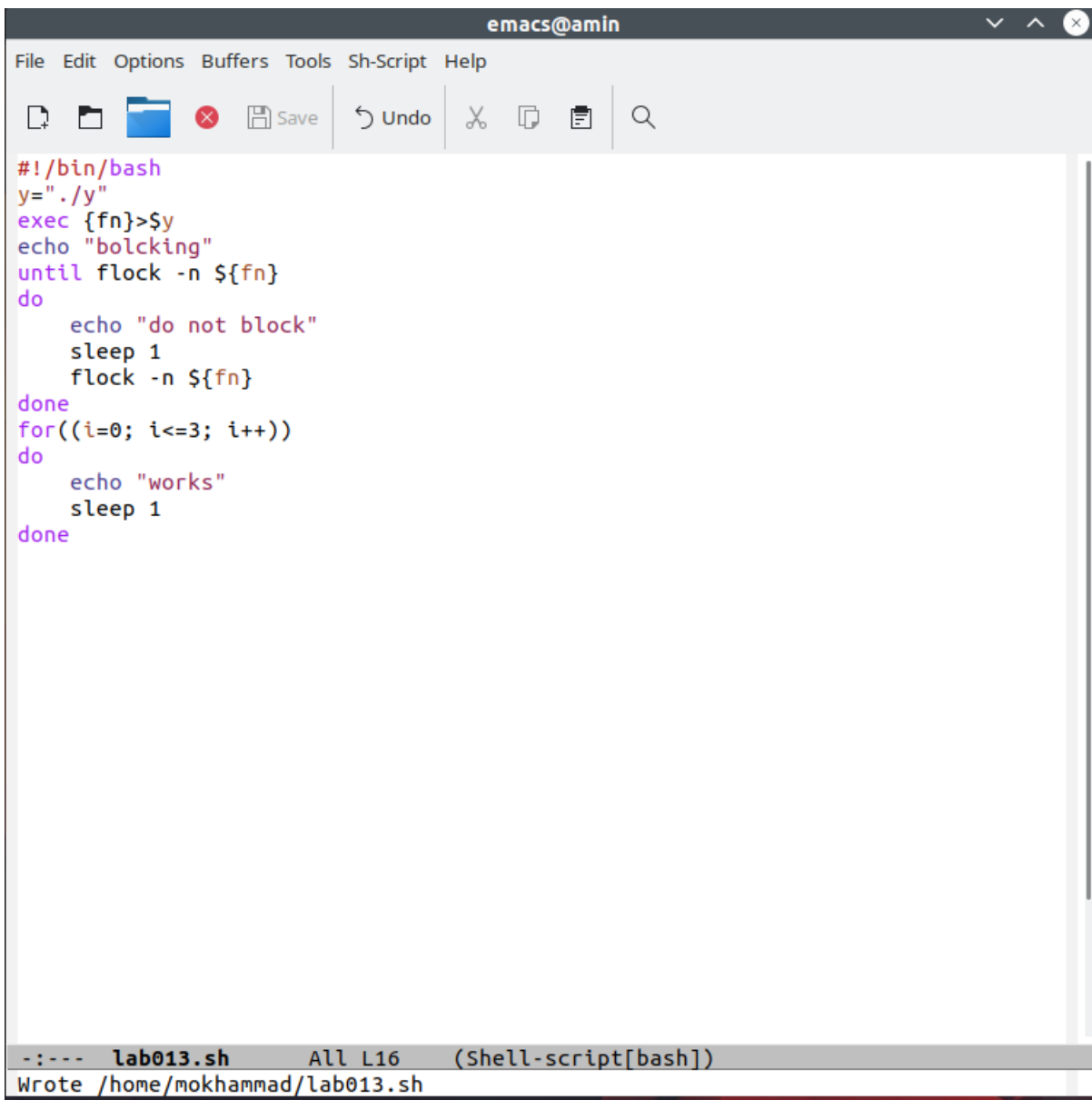
### Ход работы:

- Авторизовался в системе, создал текстовый документ, затем зашел в него.



```
mokhammad@amin: ~$ touch lab013.sh
mokhammad@amin: ~$ chmod +x lab013.sh
mokhammad@amin: ~$
```

- Мы Написали командный файл, реализующий упрощённый механизм семафоров. Ко-мандный файл должен в течение некоторого времени1дождаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени2<=>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом.



```
#!/bin/bash
y="./y"
exec {fn}>$y
echo "bolcking"
until flock -n ${fn}
do
    echo "do not block"
    sleep 1
    flock -n ${fn}
done
for((i=0; i<=3; i++))
do
    echo "works"
    sleep 1
done
```

-:--- lab013.sh All L16 (Shell-script[bash])  
Wrote /home/mokhammad/lab013.sh

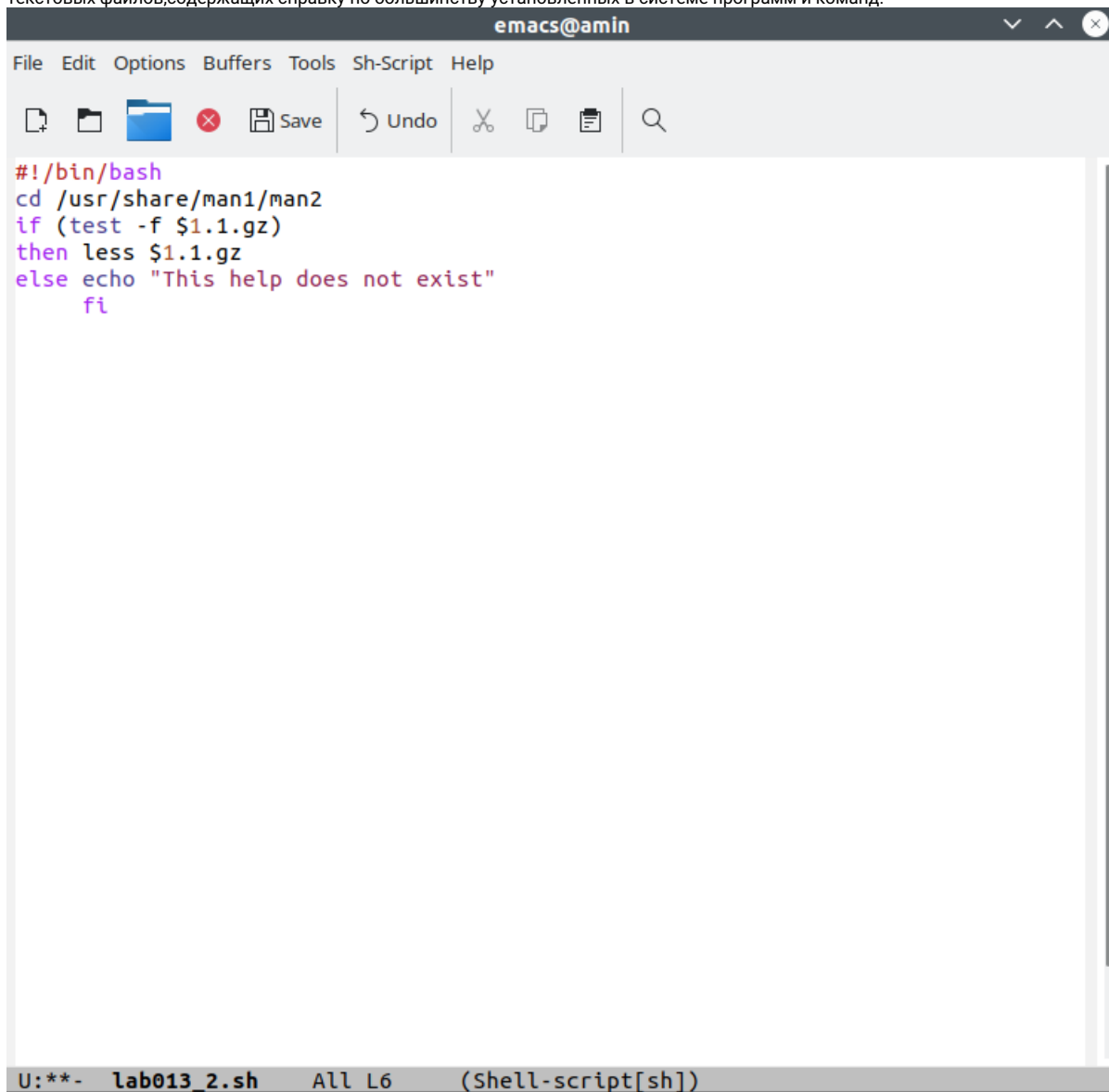
- Мы Запустили командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (> /dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы появилась возможность взаимодействия трёх и более процессов.

```
mokhammad@amin:~$ bash lab013.sh
bolcking
works
works
works
works
mokhammad@amin:~$
```

- Мы Создали текстовый файл с расширением sh. после командой chmod разрешил выполнения файла.

```
mokhammad@amin:~$ touch lab013_2.sh
mokhammad@amin:~$ chmod +x lab013_2.sh
mokhammad@amin:~$
```

- Мы Реализовали команду man с помощью командного файла. Мы Изучили содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд.



The screenshot shows the Emacs editor window titled 'emacs@amin'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The toolbar contains icons for file operations and editing. The main text area contains a shell script for the 'man' command:

```
#!/bin/bash
cd /usr/share/man1/man2
if (test -f $1.1.gz)
then less $1.1.gz
else echo "This help does not exist"
fi
```

The status bar at the bottom indicates the current file is 'lab013\_2.sh', it has 6 lines, and it is a shell script.

- Мы Запустили командный файл. Командный файл получает в виде аргумента командной строки название команды и в виде результата выдает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

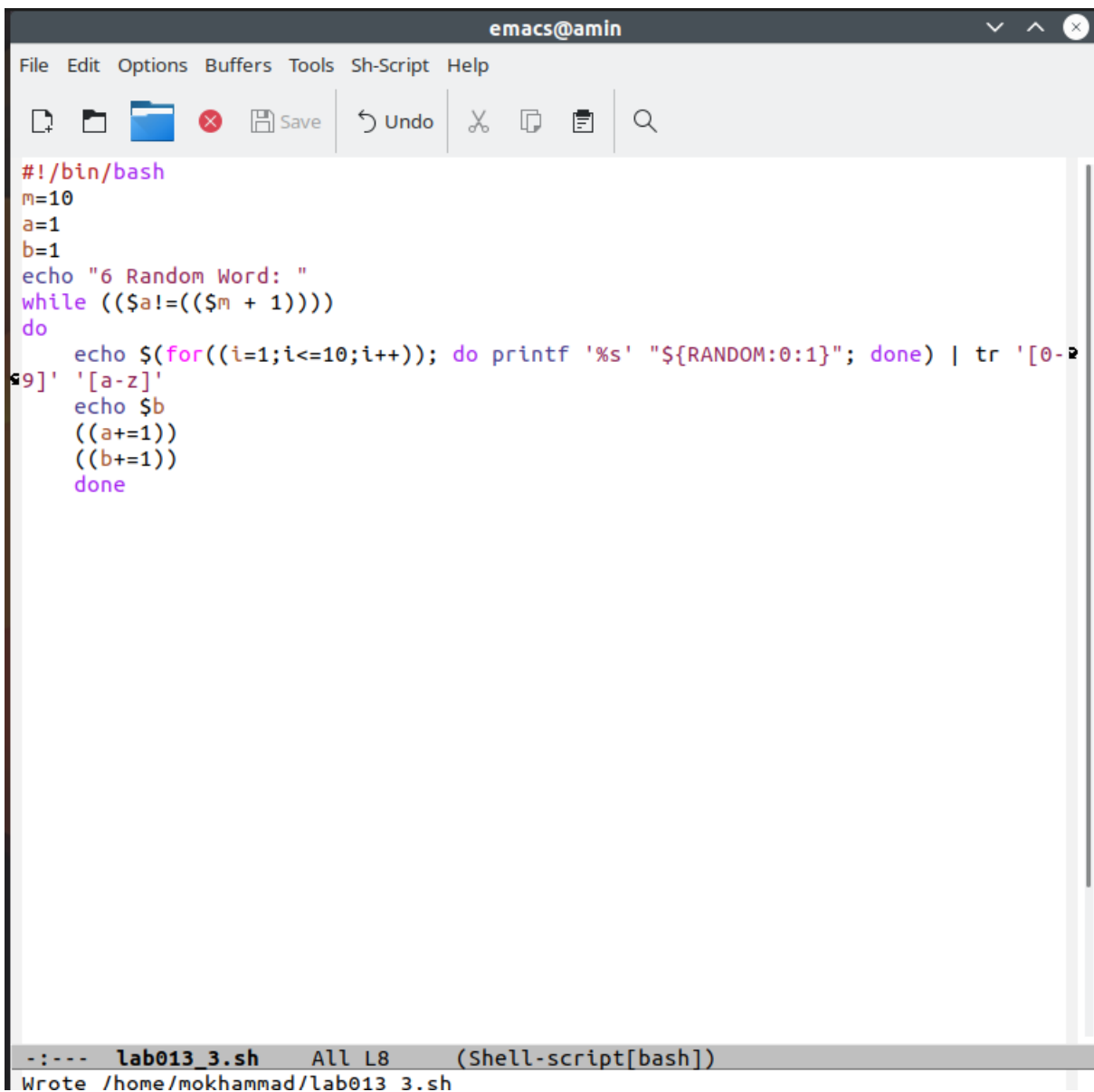
```
mokhammad@amin:~$ bash lab013_2.sh cd
This help does not exist
mokhammad@amin:~$ bash lab013_2.sh cpd
This help does not exist
mokhammad@amin:~$ bash lab013_2.sh less
mokhammad@amin:~$
```

- Каждый архив можно открыть командой less сразу же просмотрев содержимое справки.

```
mokhammad@amin: ~
.TH LESS 1 "Version 551: 11 Jun 2019"
.SH NAME
less \- opposite of more
.SH SYNOPSIS
.B "less \-?"
.br
.B "less \- \-help"
.br
.B "less \-V"
.br
.B "less \- \-version"
.br
.B "less [\-[+]aABcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX~]"
.br
.B "      [\-b \fIspace/>\fP] [\-h \fIlines/>\fP] [\-j \fIline/>\fP] [\-k \fIkeyf
ile/>\fP]"
.br
.B "      [\-o0} \fIlogfile/>\fP] [\-p \fIpattern/>\fP] [\-P \fIprompt/>\fP] [\-
t \fItag/>\fP]"
.br
.B "      [\-T \fItagsfile/>\fP] [\-x \fItab/>\fP,...] [\-y \fIlines/>\fP] [\-z]
\fIlines/>\fP]"
.br
less.1.gz
```

- Мы Создали текстовый файл с расширением .sh, после командой chmod разрешил выполнения файла.  

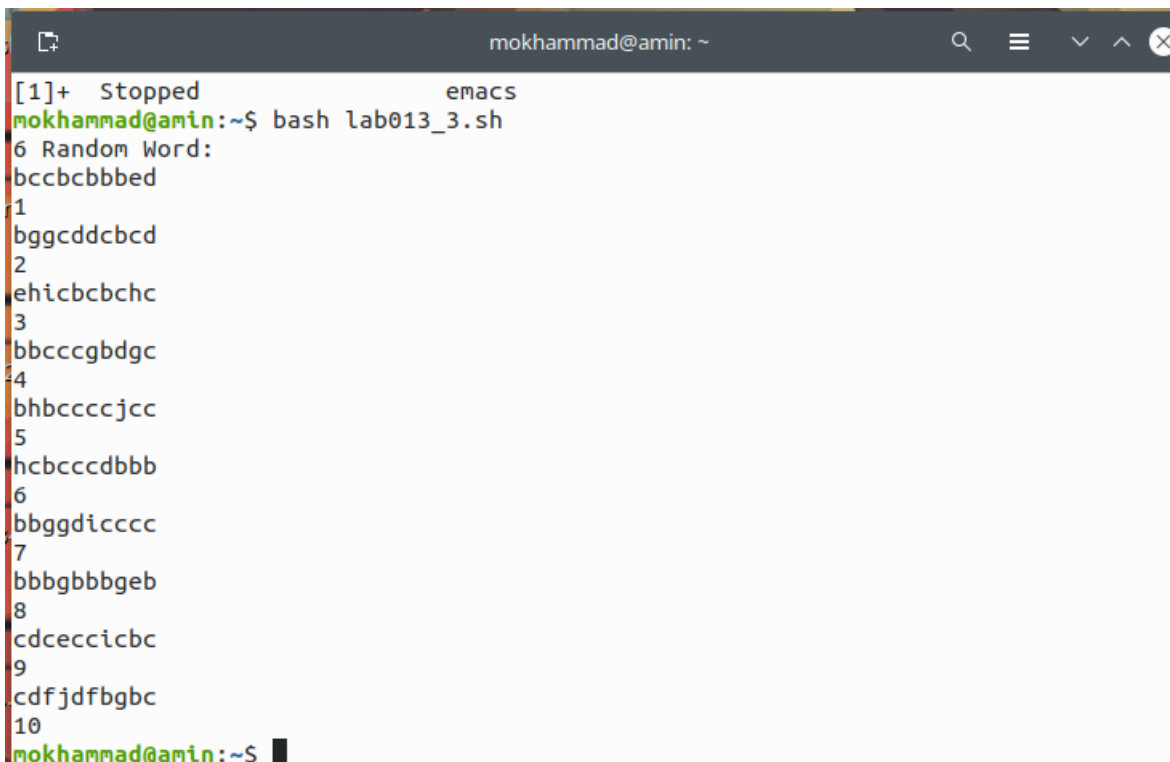
```
mokhammad@amin:~$ touch lab013_3.sh
mokhammad@amin:~$ chmod +x lab013_3.sh
mokhammad@amin:~$ emacs
```
- Используя встроенную переменную \$RANDOM, напишите, командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.



```
#!/bin/bash
m=10
a=1
b=1
echo "6 Random Word: "
while ((a!=($m + 1)))
do
    echo $(for((i=1;i<=10;i++)); do printf '%s' "${RANDOM:0:1}"; done) | tr '[:0-9:]' '[a-z]'
    echo $b
    ((a+=1))
    ((b+=1))
done
```

--:--- lab013\_3.sh All L8 (Shell-script[bash])  
Wrote /home/mokhammad/lab013\_3.sh

- мы Запустили командный файл. Как видим, вывел случайные 10 слов, состоящих из случайных букв латинского алфавита.



```
mokhammad@amin: ~  
[1]+  Stopped                  emacs  
mokhammad@amin:~$ bash lab013_3.sh  
6 Random Word:  
bccbcbbbed  
1  
bggcddcbcd  
2  
ehicbcbchc  
3  
bbcccgbdgc  
4  
bhbccccjcc  
5  
hcbcccdbbb  
6  
bbggdicccc  
7  
bbbgbbbgeb  
8  
cdceccicbc  
9  
cdfjdfbgbc  
10  
mokhammad@amin:~$
```

## Вывод:

Мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## Контрольные вопросы:

1. Найдите синтаксическую ошибку в следующей строке `while [ $1 != "exit" ] $1` следует внести в кавычки(«») `$1`
2. Как объединить (конкатенация) несколько строк в одну? С помощью знака `>|`
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать ее функционал при программировании на `bash`? Эта утилита выводит последовательность целых чисел с заданным шагом. Также можно реализовать с помощью утилиты `jot`.
4. Какой результат даст вычисление выражения `$((10/3))`? Результат: 3.
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`. В `zsh` можно настроить отдельные сочетания клавиш так, как вам нравится. Использование истории команд в `zsh` ничем особенным не отличается от `bash`. `Zsh` очень удобен для повседневной работы и делает добрую половину рутин за вас. Но стоит обратить внимание на различия между этими двумя оболочками. Например, в `zsh` после `for` обязательно вставлять пробел, нумерация массивов в `zsh` начинается с 1, чего совершенно невозможно понять. Так, если вы используете `shell` для повседневной работы, исключаяющей написание скриптов, используйте `zsh`. Если вам часто приходится писать свои скрипты, только `bash`! Впрочем, можно комбинировать. Как установить `zsh` в качестве оболочки по-умолчанию для отдельного пользователя:о
6. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))` Синтаксис верен.
7. Сравните язык `bash` с языками программирования, которые вы знаете. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?
8. Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на `си/си++`, скомпилированных с максимальной оптимизацией;
9. Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Ява-машина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам;
10. Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя большинство трансляторов других языков сценариев и подобных им по скорости исполнения программ;
11. Скорость кодов, генерируемых компилятором языка `си` фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM;
12. Скорость ассемблерных кодов `x86-64` может меньше, чем аналогичных кодов `x86`, примерно на 10%;
13. Оптимизация кодов лучше работает на процессоре Intel;
14. Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков лисп, эрланг, аук (`gawk`, `mawk`) и бэш. Разница в скорости по бэш скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32-разрядных кодах;
15. Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (`gcc`, `icc`, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром;
16. В рассматриваемых версиях `gawk`, `php`, `perl`, `bash` реализован динамический стек, позволяющий использовать всю память компьютера. Но `perl` и, особенно, `bash` используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета `ask(5,2,3)`.