

Лабораторная работа №12

НКАбд-01-22

Никита Михайлович Демидович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Контрольные вопросы	21
6	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создание директории lab11 и исполняемого файла programm1.sh	11
4.2	Исполняемый файл programm1.sh	12
4.3	Работа исполняемого файла programm1.sh	13
4.4	Содержимое директории lab12 после работы исполняемого файла programm1.sh	13
4.5	Содержимое директории /usr/share/man/man1	14
4.6	Создание исполняемого файла programm2.sh	15
4.7	Исполняемый файл programm2.sh	16
4.8	Работа исполняемого файла programm2.sh	17
4.9	Создание исполняемого файла programm3.sh	18
4.10	Исполняемый файл programm3.sh	19
4.11	Работа исполняемого файла programm3.sh	20

Список таблиц

1 Цель работы

Цель данной лабораторной 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 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на

базе оболочки Korn. Рассмотрим основные элементы программирования в оболочке `bash` (в разделе “Выполнение лабораторной работы”). В других оболочках большинство команд будет совпадать с описанными ниже.

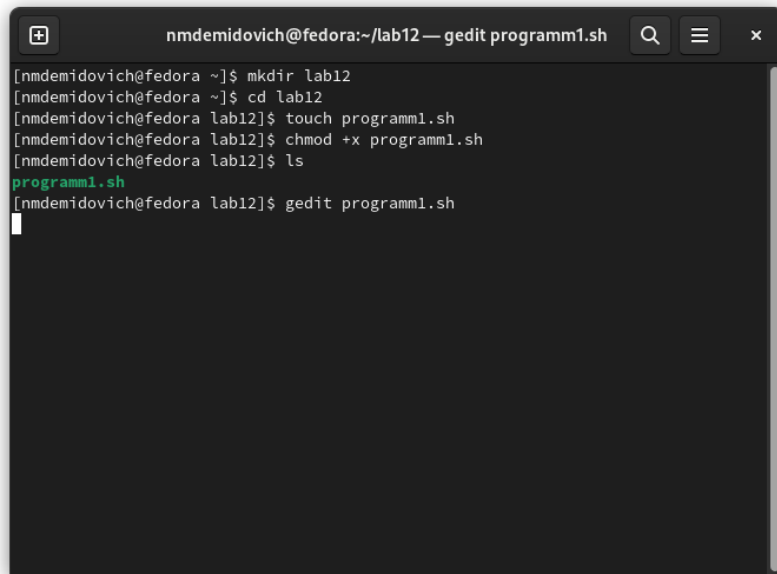
4 Выполнение лабораторной работы

На первом этапе выполнения работы я создал отдельную директорию для дальнейшей работы, командный файл `programm1.sh` и приступил к написанию командного файла, реализующего упрощённый механизм семафоров (рис. [4.1]) - (рис. [4.2]):

```
#!/bin/bash

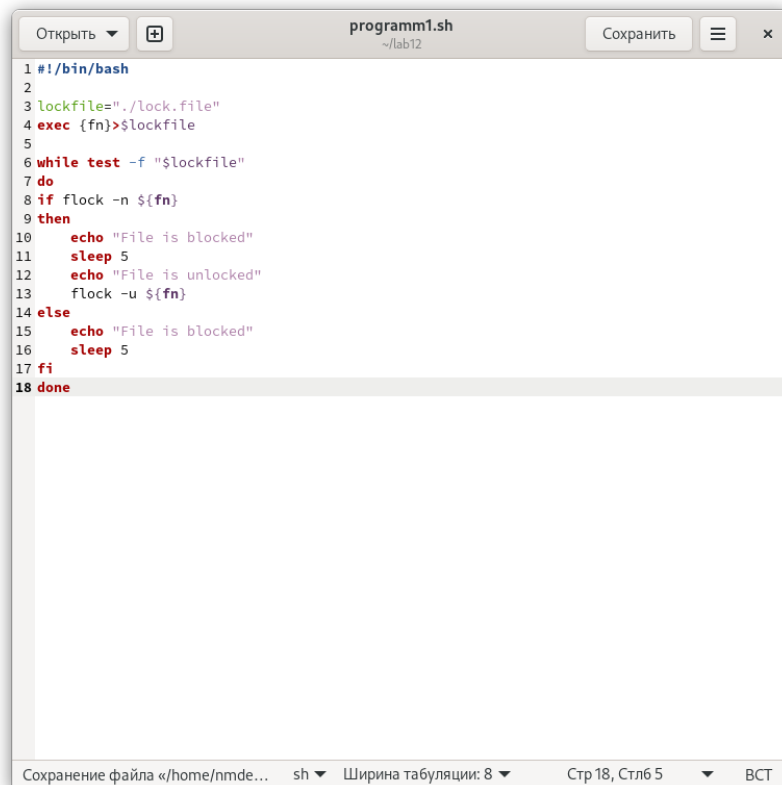
lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
if flock -n ${fn}
then
    echo "File is blocked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is blocked"
    sleep 5
fi
done
```



```
nmdemidovich@fedora:~/lab12 — gedit programm1.sh
[nmdemidovich@fedora ~]$ mkdir lab12
[nmdemidovich@fedora ~]$ cd lab12
[nmdemidovich@fedora lab12]$ touch programm1.sh
[nmdemidovich@fedora lab12]$ chmod +x programm1.sh
[nmdemidovich@fedora lab12]$ ls
programm1.sh
[nmdemidovich@fedora lab12]$ gedit programm1.sh
```

Рис. 4.1: Создание директории lab11 и исполняемого файла programm1.sh

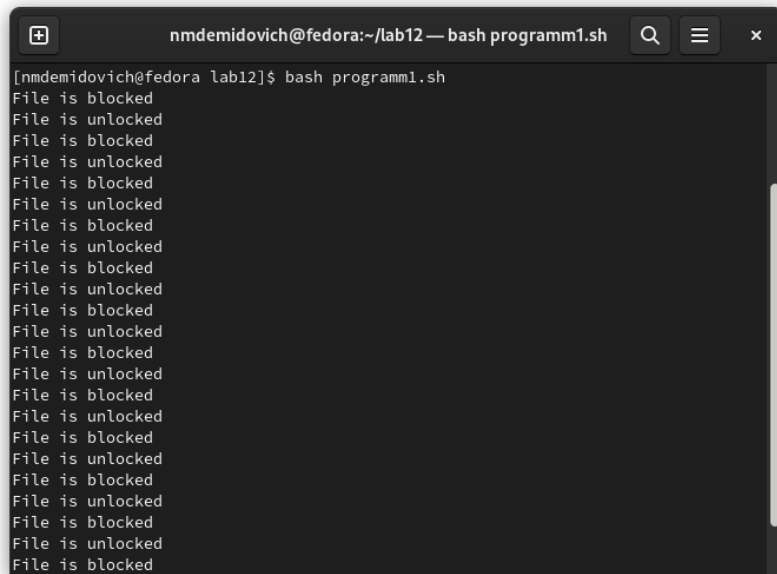


```
1 #!/bin/bash
2
3 lockfile="./lock.file"
4 exec {fn}>$lockfile
5
6 while test -f "$lockfile"
7 do
8   if flock -n ${fn}
9   then
10     echo "File is blocked"
11     sleep 5
12     echo "File is unlocked"
13     flock -u ${fn}
14   else
15     echo "File is blocked"
16     sleep 5
17   fi
18 done
```

Сохранение файла «/home/nmde... sh ▾ Ширина табуляции: 8 ▾ Стр 18, Стлб 5 ▾ ВСТ

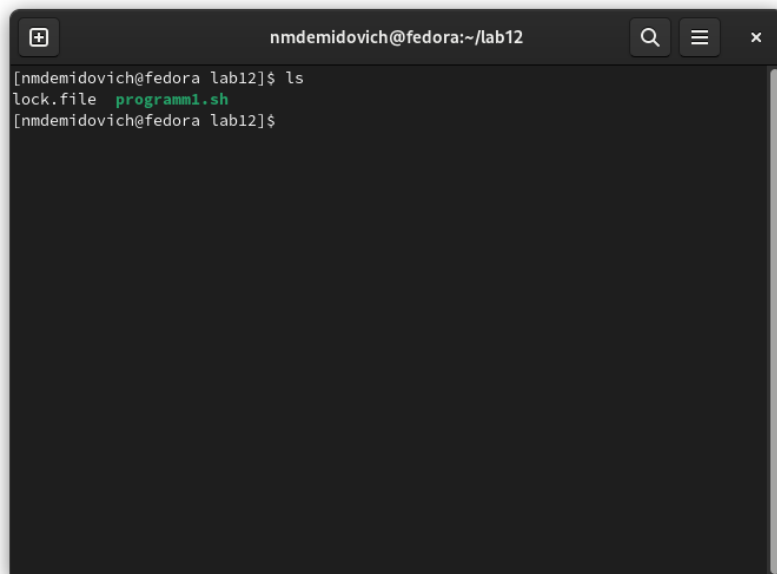
Рис. 4.2: Исполняемый файл programm1.sh

Затем я проверил работу исполняемого файла и наличие файла lock.file после работы программы (рис. [4.3]) - (рис. [4.4]):

A terminal window titled "nmdemidovich@fedora:~/lab12 — bash programm1.sh". The prompt is "[nmdemidovich@fedora lab12]\$". The user has entered "bash programm1.sh". The output is a list of 20 lines, each starting with "File is" followed by either "blocked" or "unlocked". The sequence of states is: blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked, blocked, unlocked.

```
nmdemidovich@fedora:~/lab12 — bash programm1.sh
[nmdemidovich@fedora lab12]$ bash programm1.sh
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
```

Рис. 4.3: Работа исполняемого файла programm1.sh

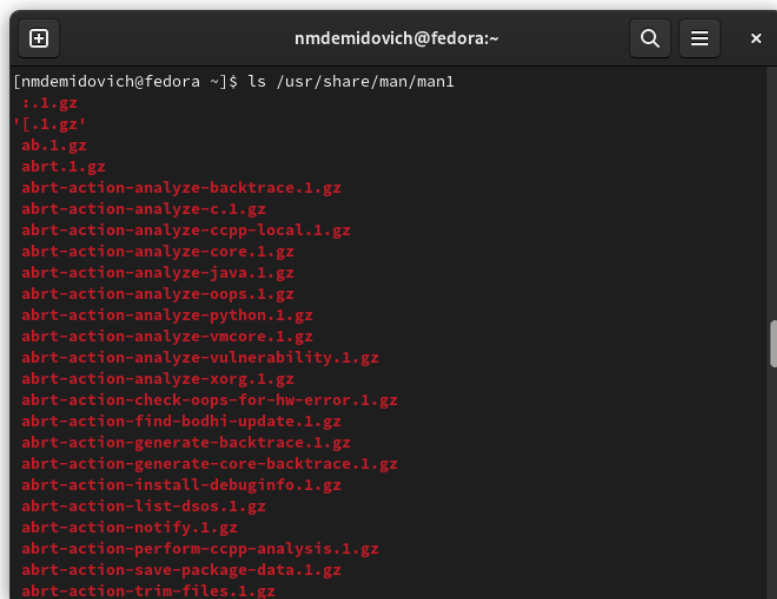
A terminal window titled "nmdemidovich@fedora:~/lab12". The prompt is "[nmdemidovich@fedora lab12]\$". The user has entered "ls". The output shows two files: "lock.file" and "programm1.sh", with "programm1.sh" highlighted in green.

```
nmdemidovich@fedora:~/lab12
[nmdemidovich@fedora lab12]$ ls
lock.file  programm1.sh
[nmdemidovich@fedora lab12]$
```

Рис. 4.4: Содержимое директории lab12 после работы исполняемого файла programm1.sh

Чтобы реализовать команду `man` с помощью командного файла, я изучил со-

держимое каталога /usr/share/man/man1 (рис. [4.5]):

A terminal window titled 'nmdemidovich@fedora:~' with search, menu, and close icons. The command 'ls /usr/share/man/man1' is executed, displaying a list of files in red text. The files include various '.1.gz' files, some with 'abrt-action-' prefixes and others with specific tool names like 'abrt', 'xorg', 'bodhi', 'dsos', and 'package-data'.

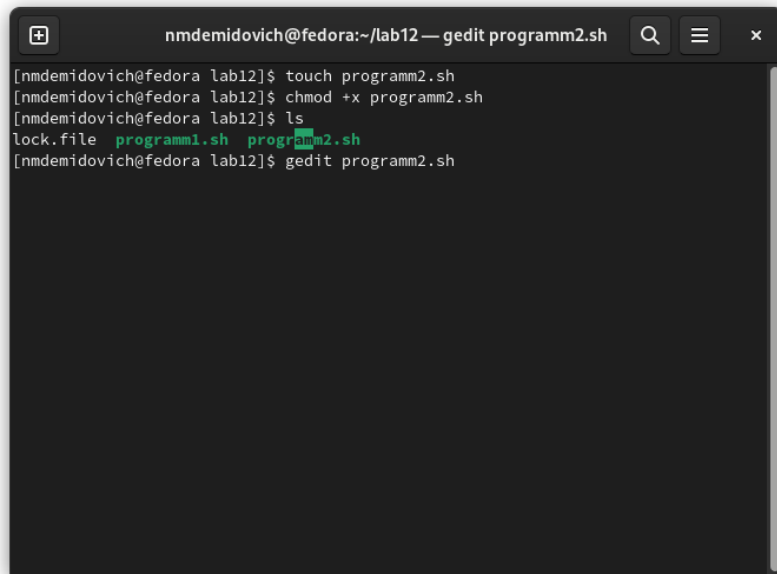
```
[nmdemidovich@fedora ~]$ ls /usr/share/man/man1
.:.1.gz
' [.1.gz'
abrt.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
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
```

Рис. 4.5: Содержимое директории /usr/share/man/man1

После этого я создал исполняемый файл для второй программы (рис. [4.6]) - (рис. [4.8]):

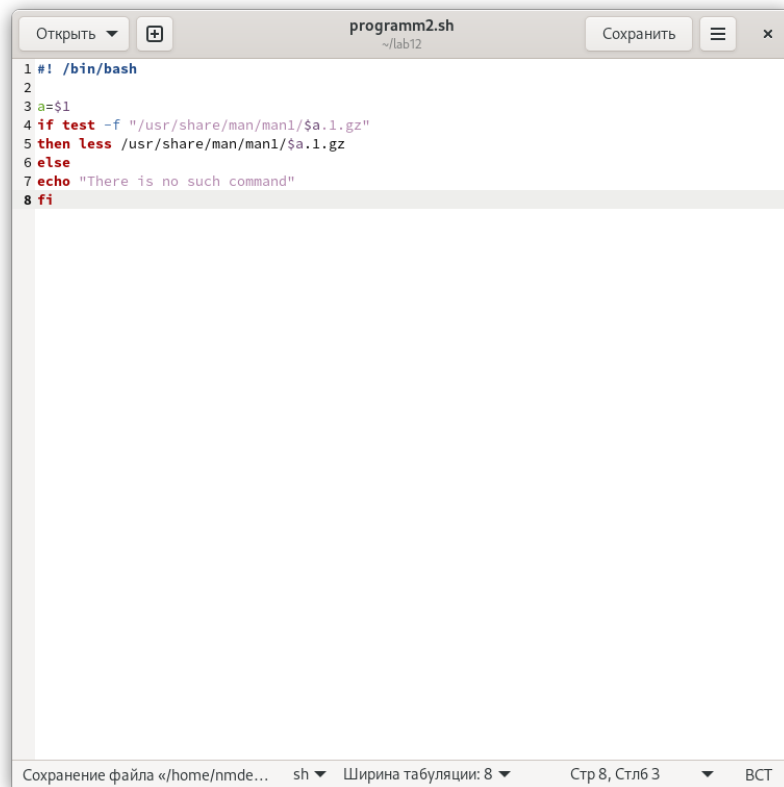
```
#!/bin/bash

a=$1
if test -f "/usr/share/man/man1/$a.1.gz"
then less /usr/share/man/man1/$a.1.gz
else
echo "There is no such command"
fi
```

A terminal window titled "nmdemidovich@fedora:~/lab12 — gedit programm2.sh". The terminal shows the following commands and output:

```
[nmdemidovich@fedora lab12]$ touch programm2.sh
[nmdemidovich@fedora lab12]$ chmod +x programm2.sh
[nmdemidovich@fedora lab12]$ ls
lock.file  programm1.sh  programm2.sh
[nmdemidovich@fedora lab12]$ gedit programm2.sh
```

Рис. 4.6: Создание исполняемого файла programm2.sh



The image shows a text editor window with the title bar "programm2.sh" and a subtitle "~lab12". The window contains a shell script with the following lines:

```
1 #! /bin/bash
2
3 a=$1
4 if test -f "/usr/share/man/man1/$a.1.gz"
5 then less /usr/share/man/man1/$a.1.gz
6 else
7 echo "There is no such command"
8 fi
```

The status bar at the bottom of the window displays the following information: "Сохранение файла «/home/nmde... sh", "Ширина табуляции: 8", "Стр 8, Стлб 3", and "ВСТ".

Рис. 4.7: Исполняемый файл programm2.sh

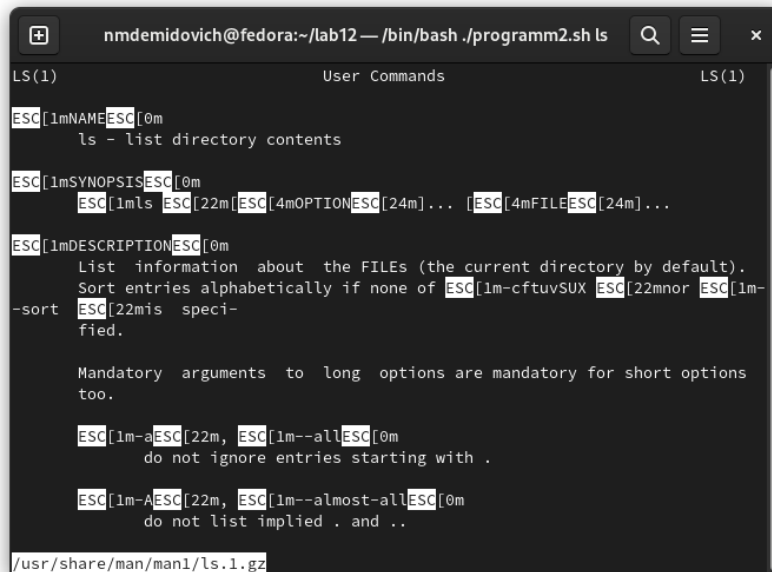


Рис. 4.8: Работа исполняемого файла programm2.sh

И на финальном этапе выполнения работы я написал командный файл programm3.sh, генерирующий случайную последовательность букв латинского алфавита (рис. [4.9]) - (рис. [4.11]):

```
#!/bin/bash
```

```
a=$1
```

```
for ((i=0; i<$a; i++))
```

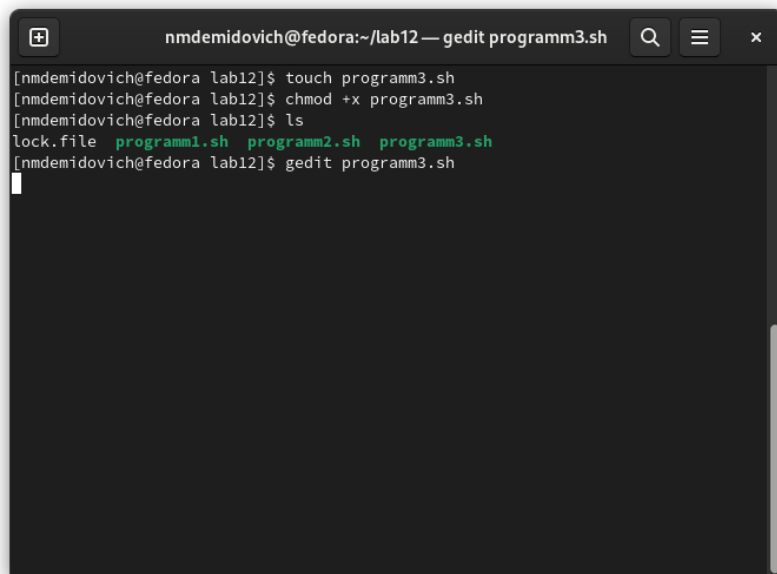
```
do
```

```
((char=$RANDOM%26+1))
```

```
case $char in
```

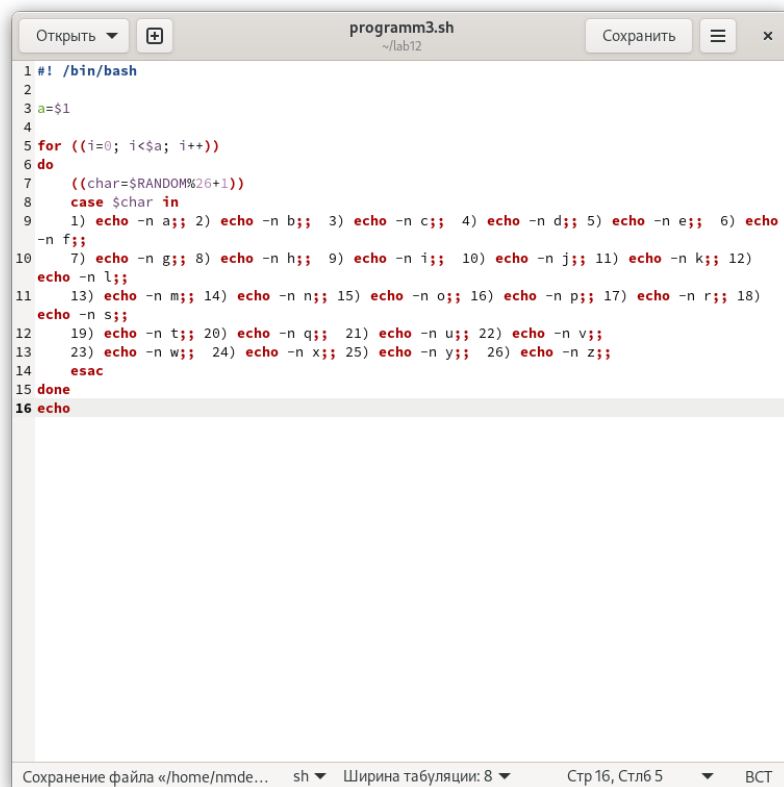
```
1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;;
7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;
13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -n r;
19) echo -n t;; 20) echo -n q;; 21) echo -n u;; 22) echo -n v;;
```

```
23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;  
esac  
done  
echo
```



```
nmdemidovich@fedora:~/lab12 — gedit programm3.sh  
[nmdemidovich@fedora lab12]$ touch programm3.sh  
[nmdemidovich@fedora lab12]$ chmod +x programm3.sh  
[nmdemidovich@fedora lab12]$ ls  
lock.file  programm1.sh  programm2.sh  programm3.sh  
[nmdemidovich@fedora lab12]$ gedit programm3.sh
```

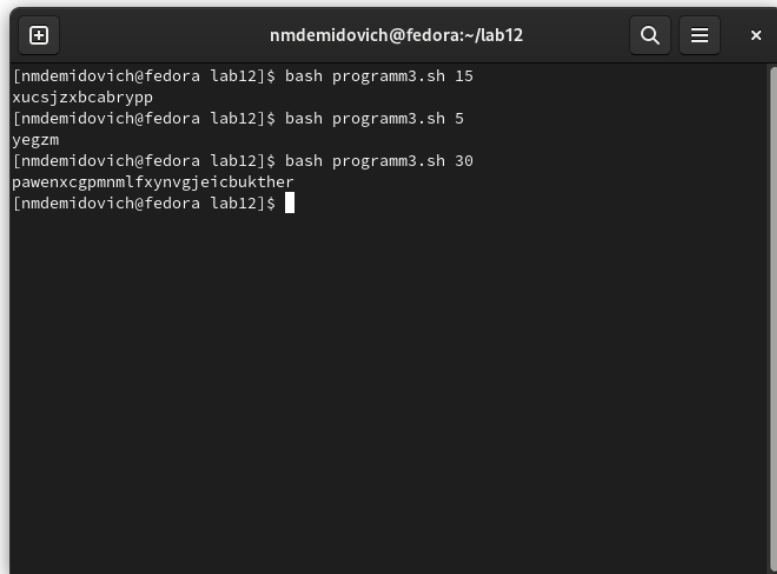
Рис. 4.9: Создание исполняемого файла programm3.sh



```
1 #! /bin/bash
2
3 a=$1
4
5 for ((i=0; i<$a; i++))
6 do
7     ((char=$RANDOM%26+1))
8     case $char in
9         1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;; 6) echo
10        -n f;;
11        7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;; 12)
12        echo -n l;;
13        13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -n r;; 18)
14        echo -n s;;
15        19) echo -n t;; 20) echo -n q;; 21) echo -n u;; 22) echo -n v;;
16        23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
17        esac
18    done
19    echo
```

Сохранение файла «/home/nmde... sh ▾ Ширина табуляции: 8 ▾ Стр 16, Стлб 5 ▾ ВСТ

Рис. 4.10: Исполняемый файл programm3.sh



```
nmdemidovich@fedora:~/lab12
[nmdemidovich@fedora lab12]$ bash programm3.sh 15
xucsjzxbcabrypp
[nmdemidovich@fedora lab12]$ bash programm3.sh 5
yegzm
[nmdemidovich@fedora lab12]$ bash programm3.sh 30
pawenxcgpmnmlfxynvgjeicbukther
[nmdemidovich@fedora lab12]$
```

Рис. 4.11: Работа исполняемого файла programm3.sh

5 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `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. Как объединить (конкатенация) несколько строк в одну?

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1="Hello," VAR2=" World" VAR3="$VAR1$VAR2" echo "$VAR3"` Результат: `Hello, World` Второй: `VAR1="Hello," VAR1+=" World" echo "$VAR1"` Результат: `Hello, World`

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?

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. Какой результат даст вычисление выражения $\$(10/3)$?

Результатом данного выражения $\$(10/3)$ будет 3, потому что это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

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. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))`

`for ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

- 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.

6 Выводы

В результате выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX и научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы

Лабораторная работа №12 (Архитектура ОС)