

Лабораторная работа № 14

**Программирование в командном процессоре ОС UNIX.
Расширенное программирование**

Юсупова Ксения Равилевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Ответы на контрольные вопросы	12
5	Выводы	16

Список иллюстраций

3.1 код для первой программы	8
3.2 проверили первый код	9
3.3 код для второй программы	9
3.4 Проверили код на работу	10
3.5 код для третьей программы	11
3.6 Проверили код на работу	11

Список таблиц

1 Цель работы

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

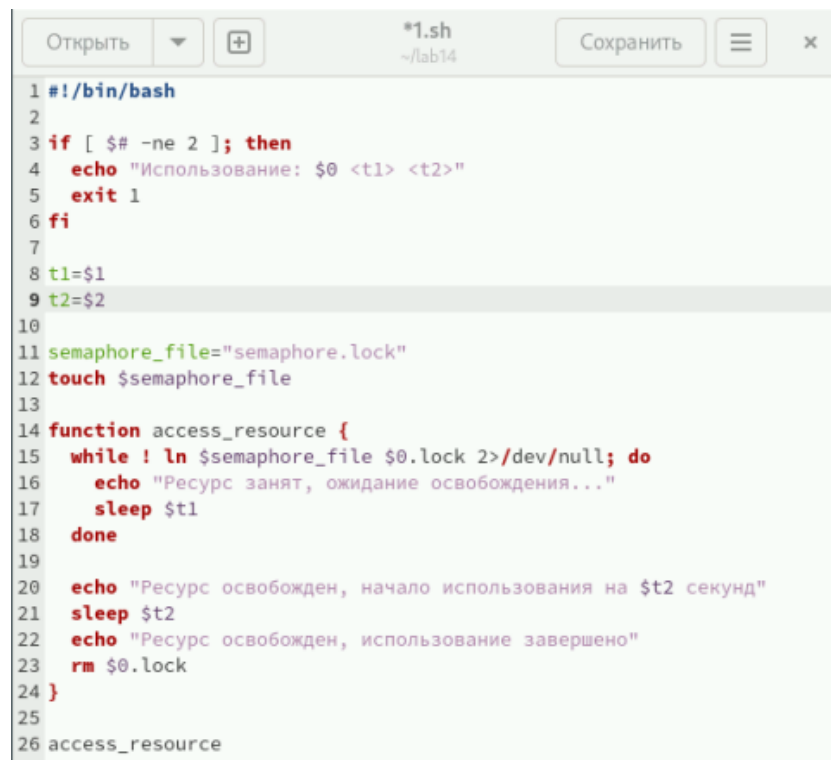
2 Задание

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

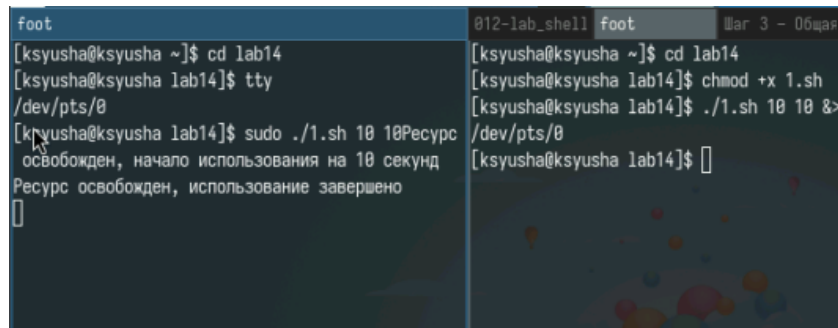
Напишем код для первой программы (рис. 3.1).



```
1 #!/bin/bash
2
3 if [ $# -ne 2 ]; then
4     echo "Использование: $0 <t1> <t2>"
5     exit 1
6 fi
7
8 t1=$1
9 t2=$2
10
11 semaphore_file="semaphore.lock"
12 touch $semaphore_file
13
14 function access_resource {
15     while ! ln $semaphore_file $0.lock 2>/dev/null; do
16         echo "Ресурс занят, ожидание освобождения..."
17         sleep $t1
18     done
19
20     echo "Ресурс освобожден, начало использования на $t2 секунд"
21     sleep $t2
22     echo "Ресурс освобожден, использование завершено"
23     rm $0.lock
24 }
25
26 access_resource
```

Рис. 3.1: код для первой программы

Проверили код на работу (рис. 3.2).

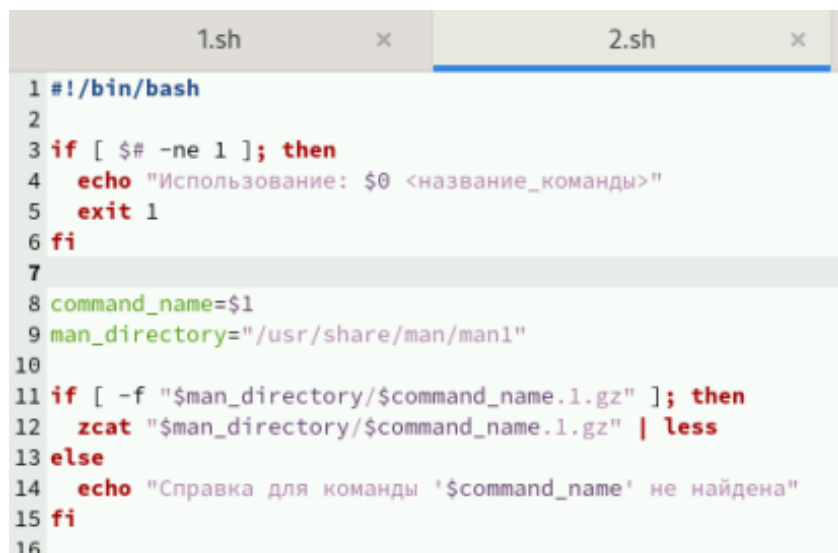


```
foot
[ksyusha@ksyusha ~]$ cd lab14
[ksyusha@ksyusha lab14]$ tty
/dev/pts/0
[ksyusha@ksyusha lab14]$ sudo ./1.sh 10 10Ресурс
освобожден, начало использования на 10 секунд
Ресурс освобожден, использование завершено
[]

012-lab_shell foot War 3 - Общая
[ksyusha@ksyusha ~]$ cd lab14
[ksyusha@ksyusha lab14]$ chmod +x 1.sh
[ksyusha@ksyusha lab14]$ ./1.sh 10 10 &
/dev/pts/0
[ksyusha@ksyusha lab14]$
```

Рис. 3.2: проверили первый код

Напишем код для второй программы (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.)(рис. 3.3).



```
1.sh x 2.sh x
1 #!/bin/bash
2
3 if [ $# -ne 1 ]; then
4     echo "Использование: $0 <название_команды>"
5     exit 1
6 fi
7
8 command_name=$1
9 man_directory="/usr/share/man/man1"
10
11 if [ -f "$man_directory/$command_name.1.gz" ]; then
12     zcat "$man_directory/$command_name.1.gz" | less
13 else
14     echo "Справка для команды '$command_name' не найдена"
15 fi
16
```

Рис. 3.3: код для второй программы

Проверили код на работу (рис. 3.4).

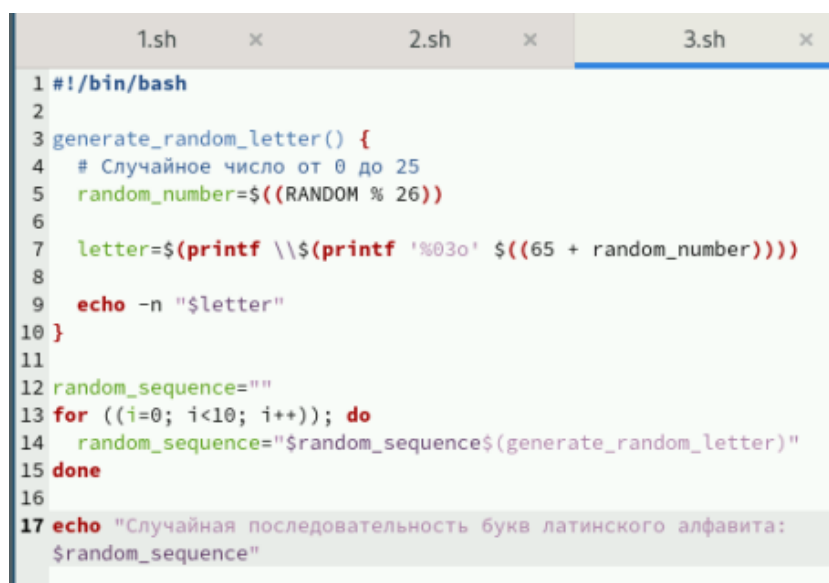
```

generate output designed for Emacs' dired mode
.TP
\fb\-f\fr
do not sort, enable \fb\-aU\fr, disable \fb\-ls\fr \fb\--color\fr
.TP
\fb\-F\fr, \fb\--classify\fr[=\fI\,WHEN\/\fr]
append indicator (one of */=>@|) to entries WHEN
.TP
\fb\--file\type\fr
likewise, except do not append '*'
.TP
\fb\--format\fr=\fI\,WORD\/\fr
across \fb\-x\fr, commas \fb\-m\fr, horizontal \fb\-x\fr, long \fb\-l\fr,
single\column \fb\-l\fr, verbose \fb\-l\fr, vertical \fb\-C\fr
.TP
\fb\--full\time\fr
like \fb\-l\fr \fb\--time\style\fr=\fI\,full\iso\/\fr
.TP
\fb\-g\fr
like \fb\-l\fr, but do not list owner
.TP
\fb\--group\directories\first\fr
group directories before files;
can be augmented with a \fb\--sort\fr option, but any
use of \fb\--sort\fr=\fI\,none\/\fr (\fb\-U\fr) disables grouping
.TP
\fb\-G\fr, \fb\--no\group\fr
in a long listing, don't print group names
.TP
\fb\-h\fr, \fb\--human\readable\fr
with \fb\-l\fr and \fb\-s\fr, print sizes like 1K 234M 2G etc.
.TP

```

Рис. 3.4: Проверили код на работу

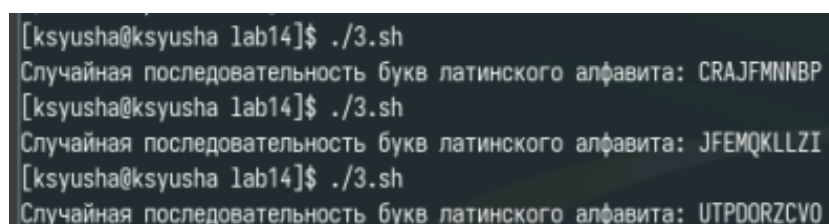
Написали код для третьей программы (Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита)(рис. 3.5).



```
1 #!/bin/bash
2
3 generate_random_letter() {
4   # Случайное число от 0 до 25
5   random_number=$((RANDOM % 26))
6
7   letter=$(printf \\$(printf '%03o' $((65 + random_number))))
8
9   echo -n "$letter"
10 }
11
12 random_sequence=""
13 for ((i=0; i<10; i++)); do
14   random_sequence="$random_sequence$(generate_random_letter)"
15 done
16
17 echo "Случайная последовательность букв латинского алфавита:
   $random_sequence"
```

Рис. 3.5: код для третьей программы

Проверили код на работу (рис. 3.6).



```
[ksyusha@ksyusha lab14]$ ./3.sh
Случайная последовательность букв латинского алфавита: CRAJFMNNBP
[ksyusha@ksyusha lab14]$ ./3.sh
Случайная последовательность букв латинского алфавита: JFEMQKLLZI
[ksyusha@ksyusha lab14]$ ./3.sh
Случайная последовательность букв латинского алфавита: UTPDORZCVO
```

Рис. 3.6: Проверили код на работу

4 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]`

Синтаксическая ошибка заключается в отсутствии пробелов после открывающей и перед закрывающей квадратными скобками. Правильный вариант:

```
bash
```

```
while [ "$1" != "exit" ]
```

Также рекомендуется заключать переменные в двойные кавычки ("`$1`"), чтобы избежать проблем, если переменная содержит пробелы или другие специальные символы.

2. Как объединить (конкатенация) несколько строк в одну?

В `bash` конкатенация строк может быть выполнена несколькими способами: •
Простое объединение:

```
bash
```

```
string1="Hello"
```

```
string2="World"
```

```
result="$string1 $string2" # result будет "Hello World"
```

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Утилита `seq` генерирует последовательность чисел. Альтернативные способы реализовать

функционал seq на bash: • Цикл for (для целочисленных последовательностей):

```
bash
```

```
for i in $(seq 1 5); do
```

```
echo $i
```

```
done
```

• Арифметический цикл for (наиболее эффективный):

```
bash
```

```
for ((i=1; i<=5; i++)); do
```

```
echo $i
```

```
done
```

• Цикл while:

```
bash
```

```
i=1
```

```
while [ $i -le 5 ]; do
```

```
echo $i
```

```
i=$((i+1))
```

```
done
```

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

Результат вычисления выражения $\$(10/3)$ в bash будет 3. Bash выполняет целочисленное деление, отбрасывая дробную часть.

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

Краткие основные отличия zsh от bash:

- **Автодополнение:** Zsh имеет значительно более развитое и настраиваемое автодополнение, чем bash.
- **Темы и внешний вид:** Zsh более гибок в настройке внешнего вида и поддерживает больше тем оформления.
- **Плагины:** Zsh имеет систему плагинов, которая позволяет расширять его функциональность.
- **Совместимость:** Bash более широко распространен и совместим с большим количеством систем.
- **Поведение по умолчанию:** Некоторые аспекты поведения по умолчанию в zsh могут отличаться от bash, что может потребовать адаптации скриптов.
- **История команд:** Zsh имеет более мощную систему истории команд.

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

Синтаксис данной конструкции верен, но для корректной работы необходимо, чтобы переменная LIMIT была определена и содержала числовое значение.

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

Сравнение bash с другими языками программирования (например, Python, C++, Java):

- **Преимущества bash:**

- **Удобство для системного администрирования:** Bash идеально подходит для автоматизации задач системного администрирования.
- **Простота и скорость для простых задач:** Bash позволяет быстро решать простые задачи без необходимости написания сложного кода.
- **Встроенная интеграция с системой:** Bash тесно интегрирован с операционной системой Linux.

- **Недостатки bash:**

- **Ограниченные возможности для сложных алгоритмов:** Bash не предназначен для разработки сложных алгоритмов.
- **Сложность отладки:** Отладка bash-скриптов может быть сложной.
- **Низкая производительность:** Bash имеет более низкую производительность по сравнению с компилируемыми языками.
- **Ограниченные возможности работы со строками:** Bash имеет ограниченные возможности для обработки строк.

В целом, bash - это мощный инструмент для автоматизации задач системного администрирования, но он не подходит для разработки сложных приложений.

5 Выводы

В ходе лабораторной работы мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.