

# **Отчет по лабораторной работе №14**

**Операционные системы**

Куокконен Дарина Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
<b>5</b>	<b>Выводы</b>	<b>15</b>
<b>6</b>	<b>Ответы на контрольные вопросы</b>	<b>16</b>

## Список иллюстраций

4.1	Создание и исполнение файла . . . . .	10
4.2	Код программы . . . . .	11
4.3	Изучение содержимого папки . . . . .	12
4.4	Код программы . . . . .	12
4.5	Исполнение программы . . . . .	13
4.6	Результат работы программы . . . . .	13
4.7	Создание и исполнение файла . . . . .	13
4.8	Код программы . . . . .	14

## 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 оболочке 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).

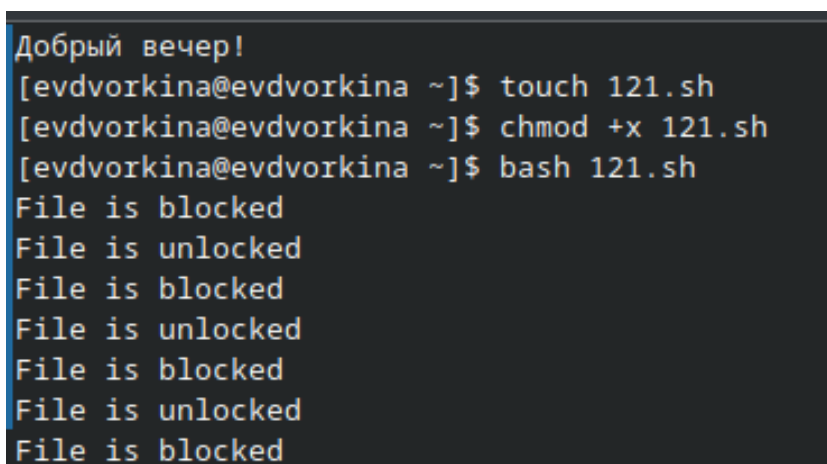
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд



будет совпадать с описанными ниже.

## 4 Выполнение лабораторной работы

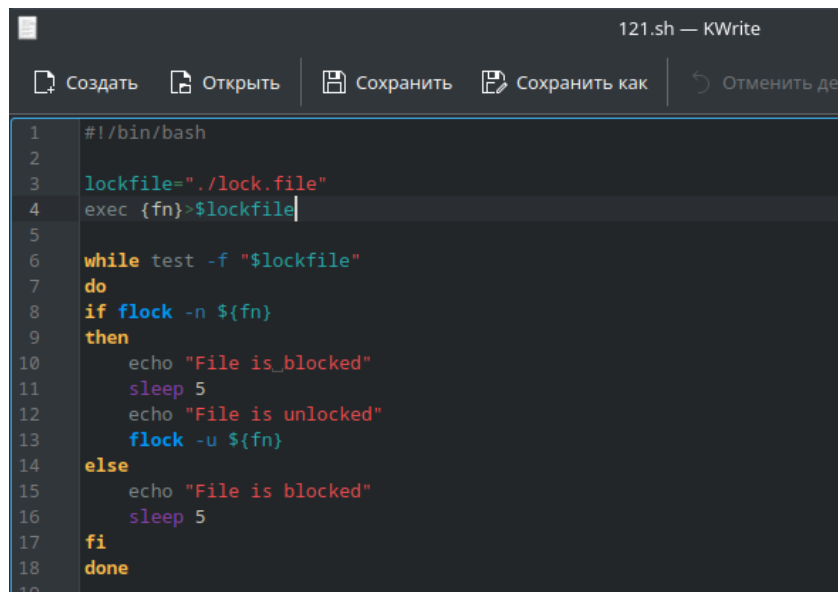
Создаю командный файл для первой программы, пишу ее, проверяю ее работу (рис. fig. 4.1).



```
Добрый вечер!  
[evdvorkina@evdvorkina ~]$ touch 121.sh  
[evdvorkina@evdvorkina ~]$ chmod +x 121.sh  
[evdvorkina@evdvorkina ~]$ bash 121.sh  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked  
File is unlocked  
File is blocked
```

Рис. 4.1: Создание и исполнение файла

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 (рис. fig. 4.2).

A screenshot of a KWrite text editor window titled "121.sh — KWrite". The window has a menu bar with options: "Создать" (Create), "Открыть" (Open), "Сохранить" (Save), "Сохранить как" (Save as), and "Отменить действие" (Undo action). The script content is as follows:

```
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
19
```

Рис. 4.2: Код программы

```
#!/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
```

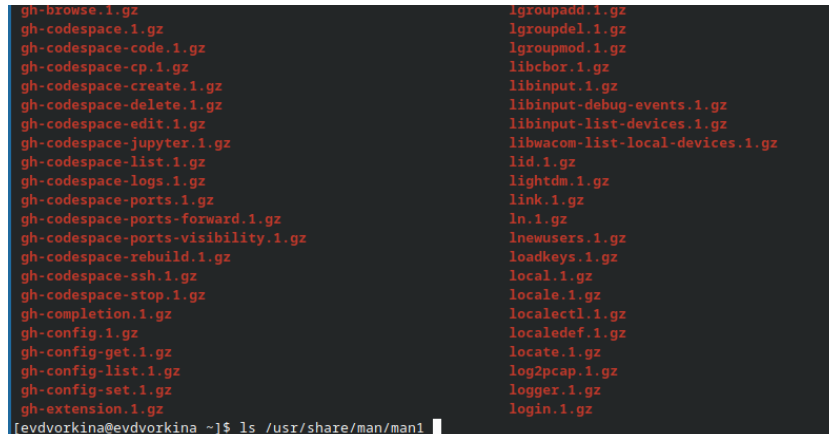
```
    echo "File is blocked"
```

```
    sleep 5
```

```
fi
```

done

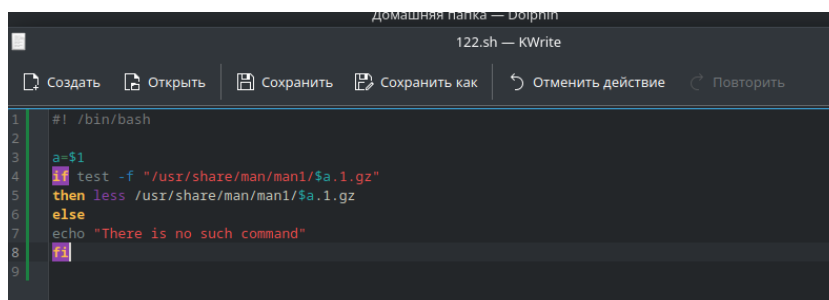
Чтобы реализовать команду `man` с помощью командного файла, изучаю содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки (рис. fig. 4.3).



```
gh-browse.1.gz
gh-codespace.1.gz
gh-codespace-code.1.gz
gh-codespace-cp.1.gz
gh-codespace-create.1.gz
gh-codespace-delete.1.gz
gh-codespace-edit.1.gz
gh-codespace-jupyter.1.gz
gh-codespace-list.1.gz
gh-codespace-logs.1.gz
gh-codespace-ports.1.gz
gh-codespace-ports-forward.1.gz
gh-codespace-ports-visibility.1.gz
gh-codespace-rebuild.1.gz
gh-codespace-ssh.1.gz
gh-codespace-stop.1.gz
gh-completion.1.gz
gh-config.1.gz
gh-config-get.1.gz
gh-config-list.1.gz
gh-config-set.1.gz
gh-extension.1.gz
lgrouppadd.1.gz
lgrouppdel.1.gz
lgrouppmod.1.gz
libcbor.1.gz
libinput.1.gz
libinput-debug-events.1.gz
libinput-list-devices.1.gz
libwacom-list-local-devices.1.gz
lid.1.gz
lightdm.1.gz
link.1.gz
ln.1.gz
lnewusers.1.gz
loadkeys.1.gz
local.1.gz
locale.1.gz
localectl.1.gz
localedef.1.gz
locate.1.gz
log2cap.1.gz
logger.1.gz
login.1.gz
[evdvorkina@evdvorkina ~]$ ls /usr/share/man/man1
```

Рис. 4.3: Изучение содержимого папки

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` (рис. fig. 4.4).



```
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
9
```

Рис. 4.4: Код программы

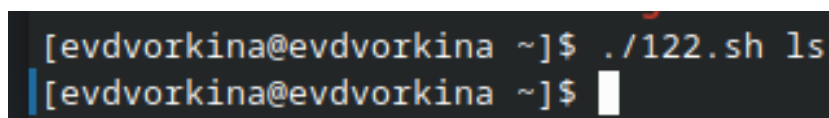
```
#!/bin/bash
```

```

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

```

Проверяю работу командного файла (рис. fig. 4.5).



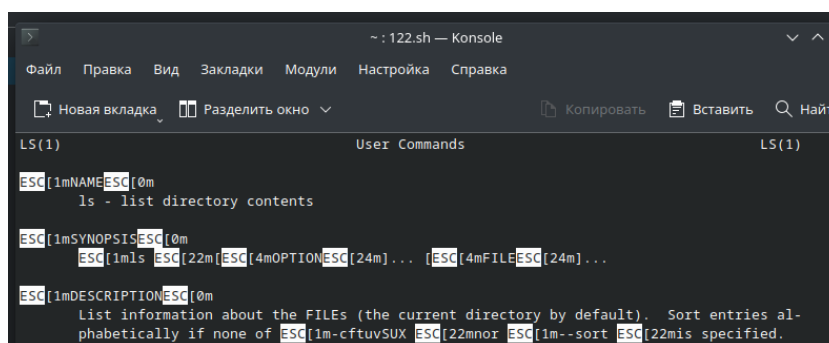
```

[evdvorkina@evdvorkina ~]$ ./122.sh ls
[evdvorkina@evdvorkina ~]$

```

Рис. 4.5: Исполнение программы

Командный файл работает так же, как и команда man, открывает справку по указанной утилите (рис. fig. 4.6).



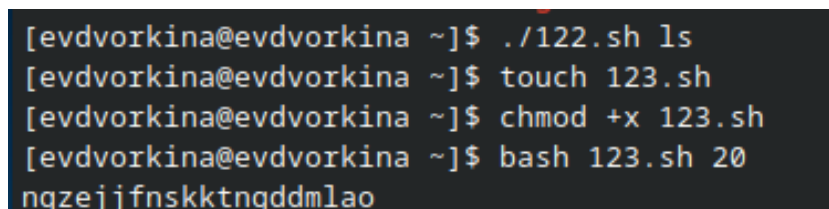
```

~:122.sh — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
LS(1)                                     User Commands                                     LS(1)
ESC[1mNAMEESC[0m
ls - list directory contents
ESC[1mSYNOPSISESC[0m
ESC[1mls ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4mFILEESC[24m]...
ESC[1mDESCRIPTIONESC[0m
List information about the FILES (the current directory by default). Sort entries al-
phabetically if none of ESC[1m-cftuvSUX ESC[22mnor ESC[1m--sort ESC[22mis specified.

```

Рис. 4.6: Результат работы программы

Создаю файл для кода третьей программы, пишу программу и проверяю ее работу (рис. fig. 4.7).



```

[evdvorkina@evdvorkina ~]$ ./122.sh ls
[evdvorkina@evdvorkina ~]$ touch 123.sh
[evdvorkina@evdvorkina ~]$ chmod +x 123.sh
[evdvorkina@evdvorkina ~]$ bash 123.sh 20
nqzejffnskktngddmlao

```

Рис. 4.7: Создание и исполнение файла

Используя встроенную переменную \$RANDOM, пишу командный файл, генерирующий случайную последовательность букв латинского алфавита. Т.к. \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767, ввожу ограничения так, чтобы была генерация чисел от 1 до 26 (рис. fig. 4.8).

A screenshot of a terminal window with a dark background. The prompt is '#! /bin/bash'. The script contains the following code:

```
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 r;; 18) echo -n s;;
    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
```

Рис. 4.8: Код программы

```
#!/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
```

## 5 Выводы

При выполнении данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX, научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 6 Ответы на 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 и недостатки 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