

# **Лабораторная работа №12**

**Дисциплина: Операционные системы**

Кабанова Варвара Дмитриевна

# Содержание

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

# List of Figures

2.1	Создание файла . . . . .	6
2.2	Скрипт №1 . . . . .	7
2.3	Проверка работы скрипта . . . . .	7
2.4	Изменённый скрипт №1 . . . . .	8
2.5	Изменённый скрипт №1 . . . . .	9
2.6	Проверка работы скрипта . . . . .	9
2.7	Реализация команды map . . . . .	10
2.8	Реализация команды map . . . . .	10
2.9	Создание файла . . . . .	10
2.10	Проверка работы скрипта . . . . .	11
2.11	Создание файла . . . . .	11
2.12	Скрипт №3 . . . . .	11

## List of Tables

# 1 Цель работы

Изучение основ программирования в оболочке ОС UNIX. Обучение написанию более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 2 Выполнение лабораторной работы

Написала 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). Для данной задачи я создала файл: `sem.sh` (рис.1) и написала соответствующий скрипт (рис.2).

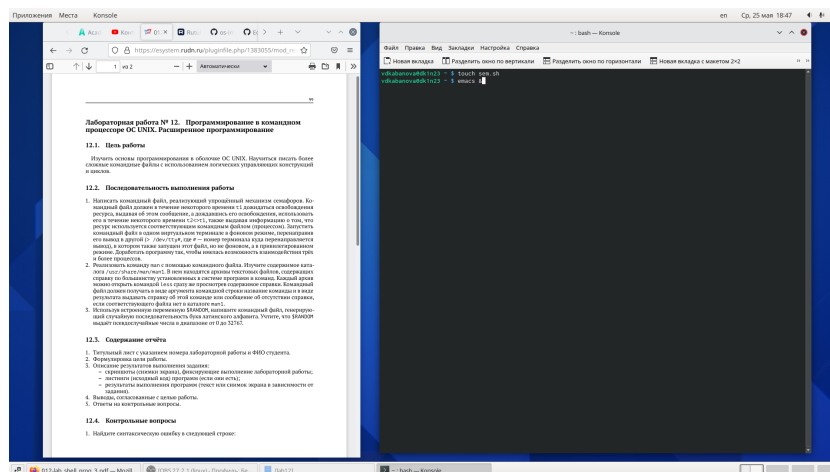


Figure 2.1: Создание файла

```

emacs@dk1n23
File Edit Options Buffers Tools Sh-Script Help

#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done

U:--- sem.sh All L23 (Shell-script[sh]) Ср мая 25 18:53 0.94
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.

U:%*- *Warnings* Bot L8 (Special) Ср мая 25 18:53 0.94
Wrote /afs/.dk.sci.pfu.edu.ru/home/v/d/vdkabanova/sem.sh

```

Figure 2.2: Скрипт №1

Далее я проверила работу написанного скрипта (команда «./sem.sh47»), предварительно добавив право на исполнение файла (команда «chmod+xsem.sh») (рис.3). Скрипт работает корректно.

```

vdkabanova@dk1n23 ~ $ touch sem.sh
vdkabanova@dk1n23 ~ $ emacs &
[1] 11724
vdkabanova@dk1n23 ~ $ chmod +x sem.sh
vdkabanova@dk1n23 ~ $ ./sem.sh 4 7
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
vdkabanova@dk1n23 ~ $

```

Figure 2.3: Проверка работы скрипта

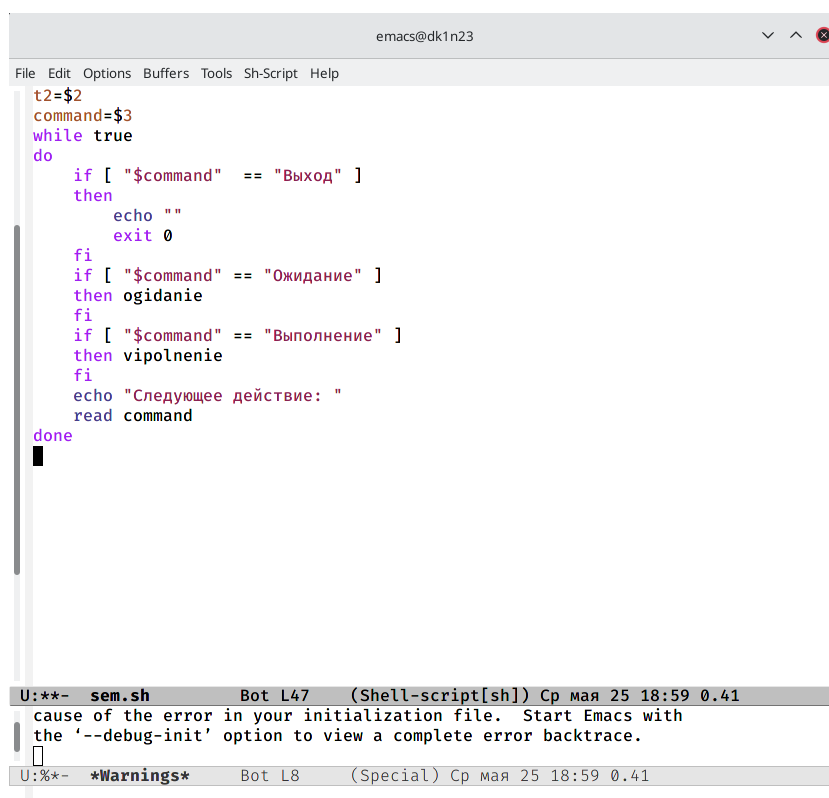
После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверила его работу (например, команда «./sem.sh2 3 Ожи-

дание > /dev/pts/1 &») (рис.4-6). Однако у меня не получилось проверить работу скрипта, так как было отказно в доступе.

Figure 2.4: Изменённый скрипт №1

8



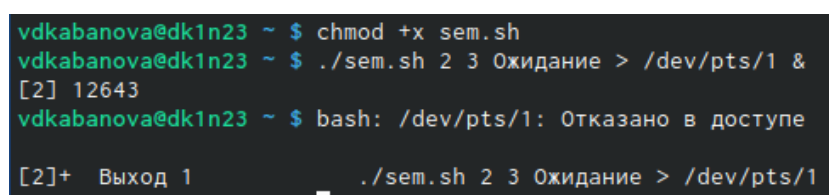


```
emacs@dk1n23
File Edit Options Buffers Tools Sh-Script Help

t2=$2
command=$3
while true
do
  if [ "$command" == "Выход" ]
  then
    echo ""
    exit 0
  fi
  if [ "$command" == "Ожидание" ]
  then ogidanie
  fi
  if [ "$command" == "Выполнение" ]
  then vipolnenie
  fi
  echo "Следующее действие: "
  read command
done

U:*** sem.sh Bot L47 (Shell-script[sh]) Ср мая 25 18:59 0.41
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
U:%*- *Warnings* Bot L8 (Special) Ср мая 25 18:59 0.41
```

Figure 2.5: Изменённый скрипт №1



```
vdkabanova@dk1n23 ~ $ chmod +x sem.sh
vdkabanova@dk1n23 ~ $ ./sem.sh 2 3 Ожидание > /dev/pts/1 &
[2] 12643
vdkabanova@dk1n23 ~ $ bash: /dev/pts/1: Отказано в доступе

[2]+  Выход 1 ./sem.sh 2 3 Ожидание > /dev/pts/1
```

Figure 2.6: Проверка работы скрипта

Реализовала команду man с помощью командного файла. Изучила содержимое каталога /usr/share/man/man1 (рис.7-8). 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.

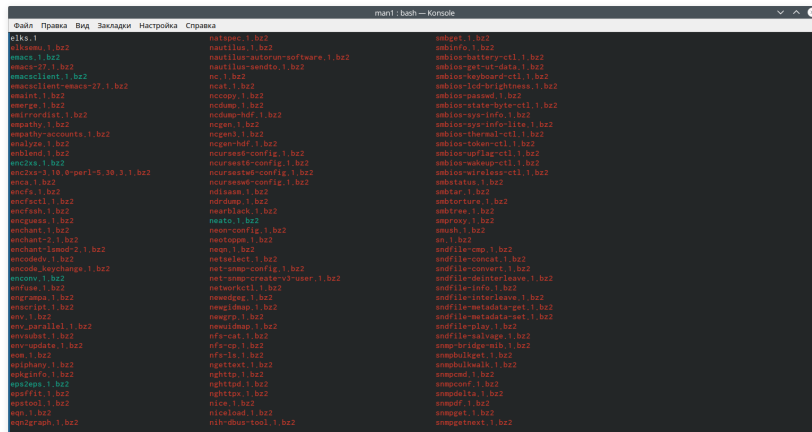


Figure 2.7: Реализация команды man

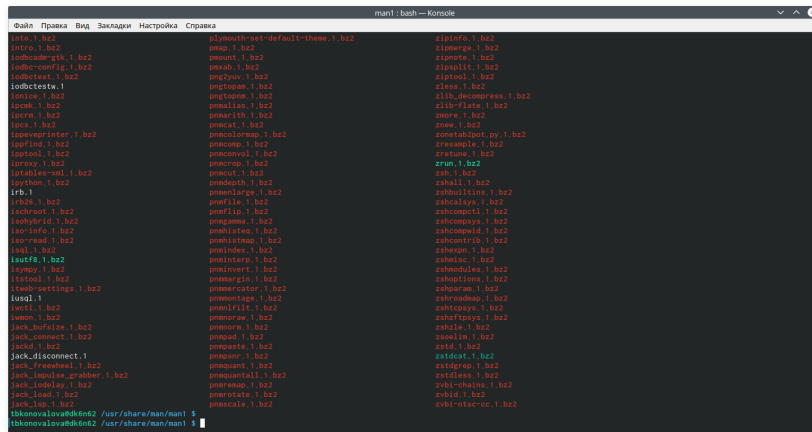


Figure 2.8: Реализация команды man

Для данной задачи я создала файл: man.sh (рис.9) и написала соответствующий скрипт (рис.10).

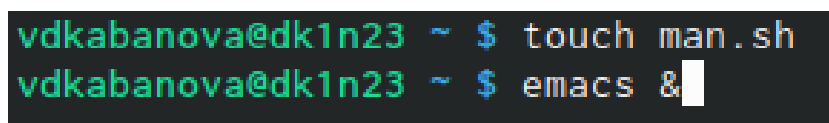


Figure 2.9: Создание файла

## Скрипт №2

Далее я проверила работу написанного скрипта (команды «./man.shls» и «./man.sh mkdir»), предварительно добавив право на исполнение файла (команда

«chmod +x man.sh») (рис.11). Скрипт работает корректно.

```
vdkabanova@dk1n23 ~ $ chmod +x man.sh
vdkabanova@dk1n23 ~ $ ./man.sh ls
Справки по данной команде нет
vdkabanova@dk1n23 ~ $ ./man.sh mkdir
Справки по данной команде нет
vdkabanova@dk1n23 ~ $
```

Figure 2.10: Проверка работы скрипта

Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создала файл: random.sh (рис.12) и написала соответствующий скрипт (рис.13).

```
Справки по данной команде нет
vdkabanova@dk1n23 ~ $ touch random.sh
vdkabanova@dk1n23 ~ $ emacs &
```

Figure 2.11: Создание файла

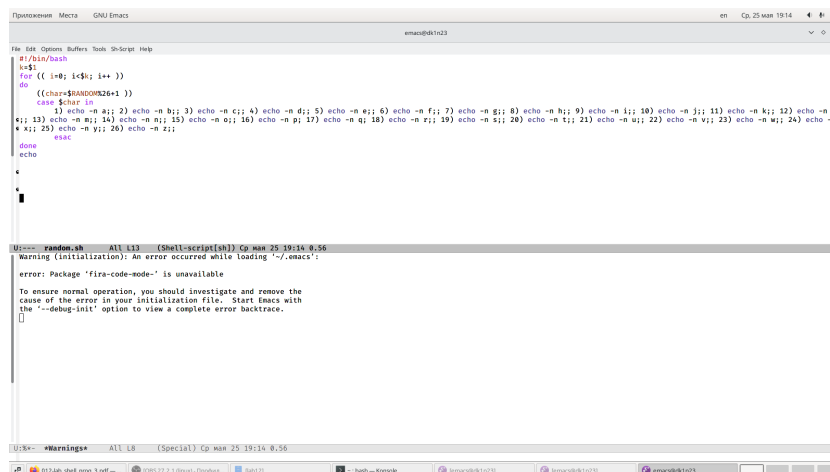


Figure 2.12: Скрипт №3

Далее я проверила работу написанного скрипта (команды «./random.sh 7» и «./random.sh 15»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh»). Скрипт работает корректно.

### 3 Ответы на 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. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:

- Первый:

```
VAR1="Hello,  
"VAR2=" World"  
VAR3="VAR1VAR2"  
echo "$VAR3"  
Результат: Hello, World
```

- Второй:

```
VAR1="Hello,"  
VAR1+=" World"  
echo "$VAR1"  
Результат: 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` 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 данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().
7. 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.

## 4 Выводы

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