

Лабораторная работа No 12.

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

Боровиков Даниил Александрович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	12
4	Контрольные вопросы	13

Список иллюстраций

2.1	Написание скрипта lab12_1.sh	7
2.2	Право на выполнение и запуск lab12_1.sh	7
2.3	Написание скрипта lab12_2.sh	8
2.4	Право на выполнение и запуск lab12_2.sh	8
2.5	Написание скрипта lab12_3.sh	9
2.6	Право на выполнение и запуск lab12_3.sh	10
2.7	Написание скрипта lab12_4.sh	10
2.8	Право на выполнение и запуск lab12_4.sh	11

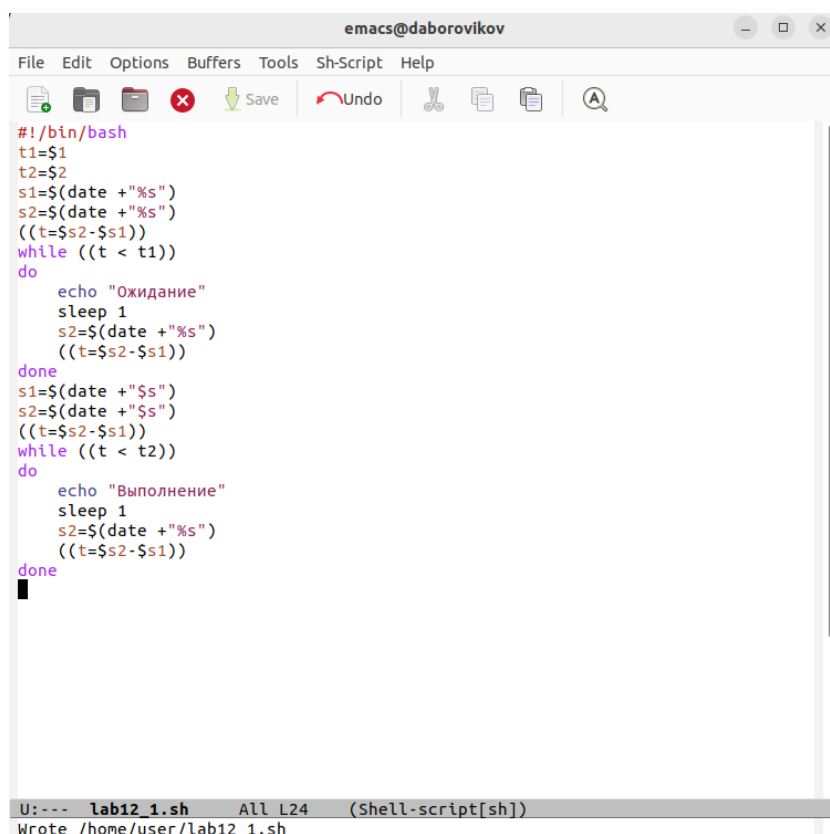
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

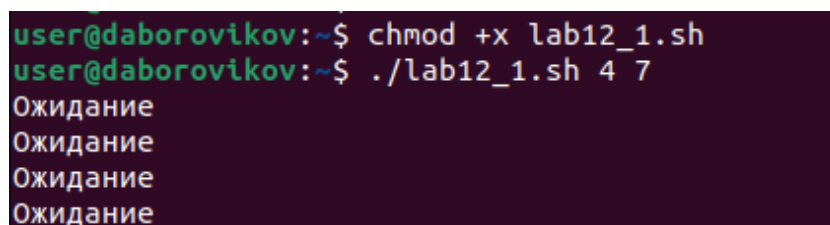
Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, 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. (рис. fig. 2.1).



```
emacs@daborovikov
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/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:--- lab12_1.sh All L24 (Shell-script[sh])
Wrote /home/user/lab12_1.sh
```

Рис. 2.1: Написание скрипта lab12_1.sh

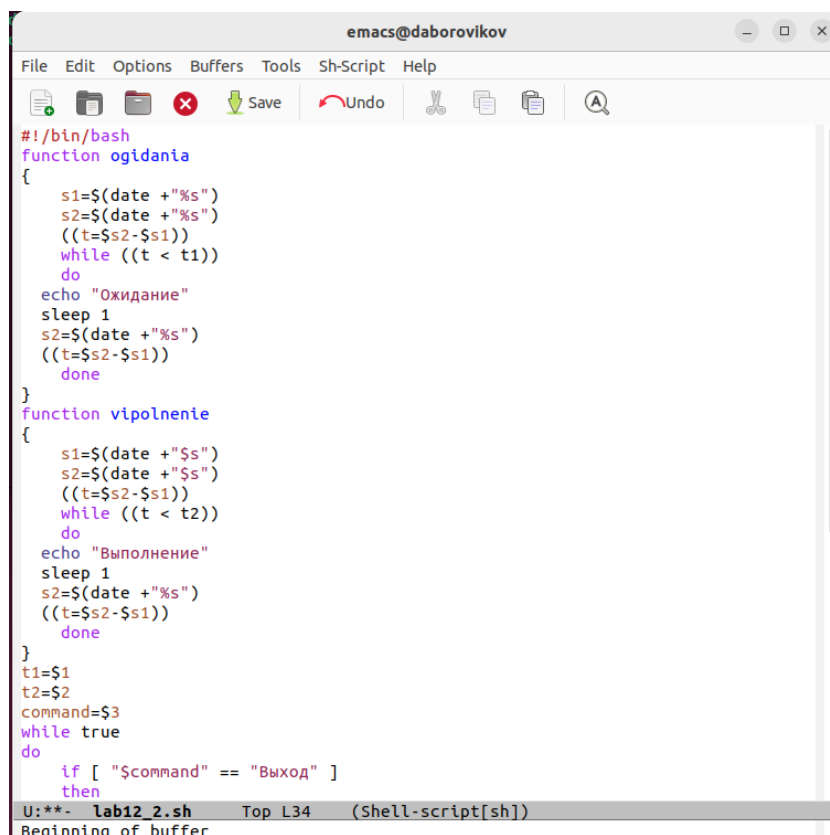
В терминале дадим файлу право на исполнение. Запустим файл и проверим.(рис. fig. 2.2).



```
user@daborovikov:~$ chmod +x lab12_1.sh
user@daborovikov:~$ ./lab12_1.sh 4 7
Ожидание
Ожидание
Ожидание
Ожидание
```

Рис. 2.2: Право на выполнение и запуск lab12_1.sh

Доработаем программу так, чтобы имелась возможность взаимодействия трёх и более процессов.(рис. fig. 2.3).



```
#!/bin/bash
function ogidania
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-s1))
    while ((t < t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s")
        ((t=s2-s1))
    done
}
function vipolnenie
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-s1))
    while ((t < t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s")
        ((t=s2-s1))
    done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        break
    fi
    ogidania
    vipolnenie
done
```

Рис. 2.3: Написание скрипта lab12_2.sh

В терминале дадим файлу право на исполнение. Запустим файл и проверим.(рис. fig. 2.4).



```
user@daborovikov: ~
user@daborovikov:~$ ./lab12_2.sh 2 3 Ожидание > /dev/pts/1 &
[1] 10211
user@daborovikov:~$ bash: /dev/pts/1: Отказано в доступе

[1]+  Выход 1                  ./lab12_2.sh 2 3 Ожидание > /dev/pts/1
user@daborovikov:~$ ./lab12_2.sh 2 3 Ожидание > /dev/pts/1 &
[1] 10811
user@daborovikov:~$ bash: /dev/pts/1: Отказано в доступе

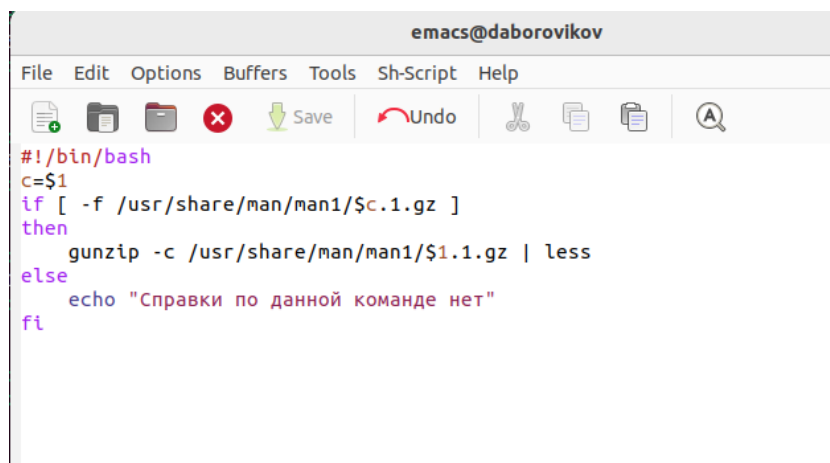
[1]+  Выход 1                  ./lab12_2.sh 2 3 Ожидание > /dev/pts/1
user@daborovikov:~$ ./lab12_2.sh 2 3 Ожидание > /dev/pts/2 &
[1] 11096
user@daborovikov:~$ bash: /dev/pts/2: Отказано в доступе

[1]+  Выход 1                  ./lab12_2.sh 2 3 Ожидание > /dev/pts/2
user@daborovikov:~$
```

Рис. 2.4: Право на выполнение и запуск lab12_2.sh

Реализуем команду man с помощью командного файла. Изучите содержимое

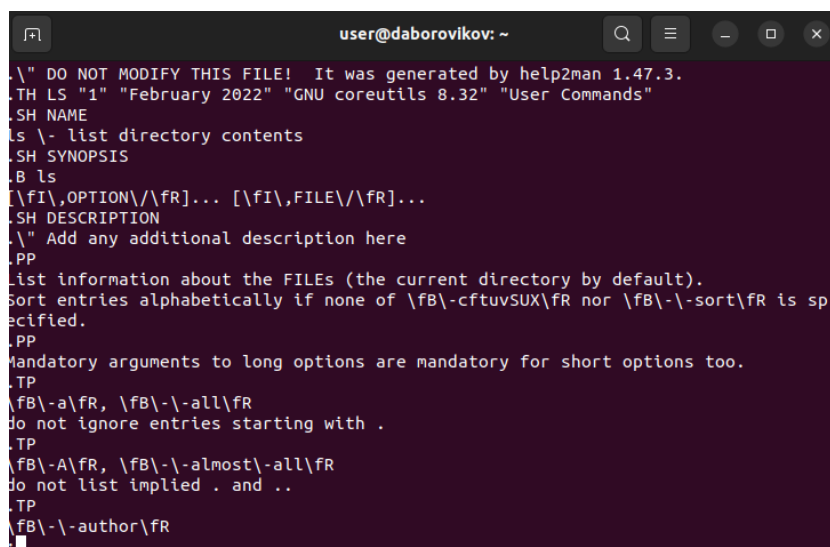
ката-лога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.(рис. fig. 2.5).

The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@daborovikov". Below the title bar is a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Underneath the menu bar is a toolbar with icons for file operations (new, open, save, delete), editing (undo, redo), and search. The main text area contains a shell script written in a syntax-highlighted format. The script starts with a shebang line, followed by a variable assignment, an if-statement that checks for the existence of a gzipped manual file and either unzips and pipes it to less, or echoes a message in Russian if the file is not found. The script ends with a fi statement.

```
#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/${c}.1.gz ]
then
    gunzip -c /usr/share/man/man1/${c}.1.gz | less
else
    echo "Справки по данной команде нет"
fi
```

Рис. 2.5: Написание скрипта lab12_3.sh


В терминале дадим файлу право на исполнение. Запускаем файл и проверяем.(рис. fig. 2.6).



```
user@daborovikov: ~  
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.  
TH LS "1" "February 2022" "GNU coreutils 8.32" "User Commands"  
SH NAME  
ls \- list directory contents  
SH SYNOPSIS  
B ls  
[\\FI\\,OPTION\\/\fR]... [\\FI\\,FILE\\/\fR]...  
SH DESCRIPTION  
.\" Add any additional description here  
PP  
List information about the FILES (the current directory by default).  
Sort entries alphabetically if none of \\fB\\-cftuvSUX\\fR nor \\fB\\-\\-sort\\fR is sp  
ecified.  
PP  
Mandatory arguments to long options are mandatory for short options too.  
TP  
\\fB\\-a\\fR, \\fB\\-\\-all\\fR  
do not ignore entries starting with .  
TP  
\\fB\\-A\\fR, \\fB\\-\\-almost\\-all\\fR  
do not list implied . and ..  
TP  
\\fB\\-\\-author\\fR
```

Рис. 2.6: Право на выполнение и запуск lab12_3.sh

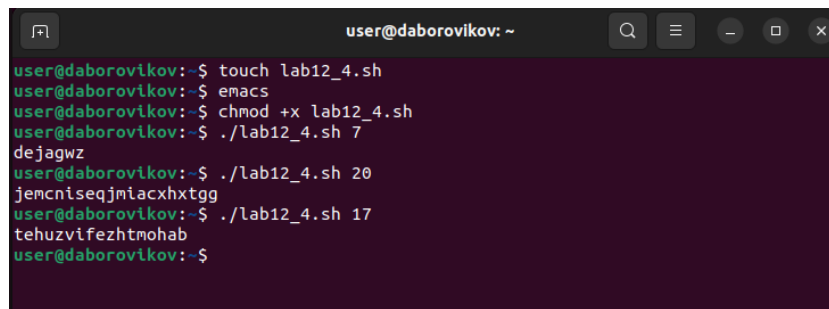
Используя встроенную переменную \$RANDOM, напомним командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.(рис. fig. 2.7).



```
emacs@daborovikov  
File Edit Options Buffers Tools Sh-Script Help  
a1/b1n/bash  
#!/bin/bash  
s=$1  
for (( i=0; i<$s; 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 q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;;  
        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. 2.7: Написание скрипта lab12_4.sh

В терминале дадим файлу право на исполнение. Запускаем файл. (рис. fig. 2.8).



```
user@daborovikov: ~  
user@daborovikov:~$ touch lab12_4.sh  
user@daborovikov:~$ emacs  
user@daborovikov:~$ chmod +x lab12_4.sh  
user@daborovikov:~$ ./lab12_4.sh 7  
dejagwz  
user@daborovikov:~$ ./lab12_4.sh 20  
jemcniseqjmiacxhxtgg  
user@daborovikov:~$ ./lab12_4.sh 17  
tehuzvifezhtmohab  
user@daborovikov:~$
```

Рис. 2.8: Право на выполнение и запуск lab12_4.sh

3 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.

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

```
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="VAR1VAR2" 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` 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++)) 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.