

# **Отчёт по лабораторной работе 12**

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

Ангелина Дмитриевна Чванова

# 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>9</b>
<b>5</b>	<b>Выводы</b>	<b>14</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>15</b>

## Список иллюстраций

4.1	Создание файла, открытие emacs в фоновом режиме. Команда chmod +x, которая дает право на исполнение. . . . .	9
4.2	Скрипт программы . . . . .	10
4.3	Скрипт программы . . . . .	11
4.4	работа программы . . . . .	11
4.5	Создание файла, открытие emacs в фоновом режиме. Команда chmod +x, которая дает право на исполнение. . . . .	12
4.6	Скрипт программы . . . . .	12
4.7	Создание файла, открытие emacs в фоновом режиме. Команда chmod +x, которая дает право на исполнение. . . . .	13
4.8	Скрипт программы . . . . .	13
4.9	работа программы . . . . .	13

## 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 Теоретическое введение

Выполнение условного оператора `if` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `if`. Затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `then`. Фраза `elif` проверяется в том случае, когда предыдущая проверка была ложной. Строка, содержащая служебное слово `else`, является необязательной. Если она присутствует, то последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `else`, будет выполнена только при условии, что последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `if` или `elif`, возвращает ненулевой код завершения (ложь).



## 4 Выполнение лабораторной работы

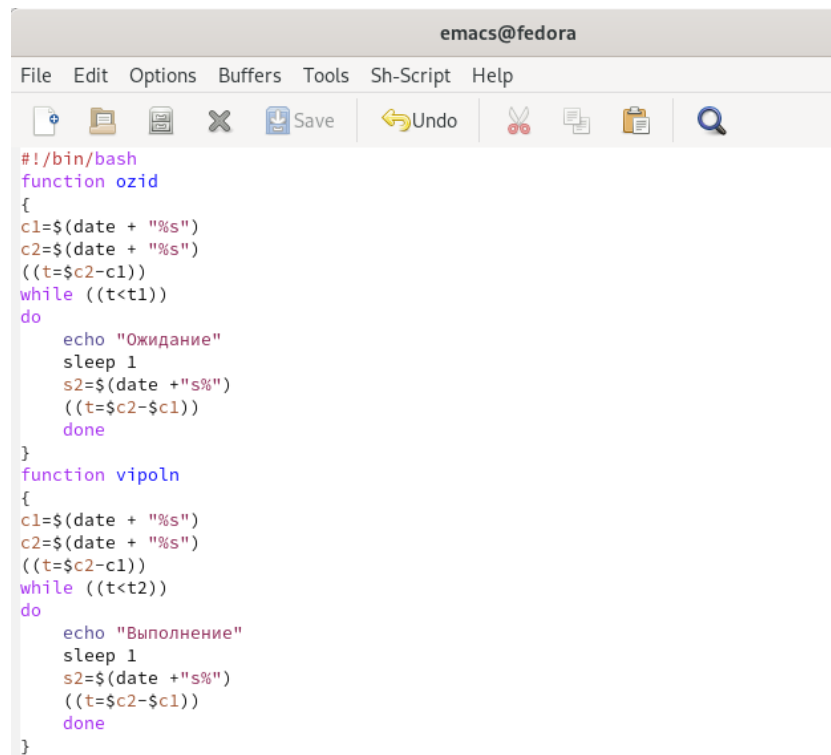
1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени  $t_1$  дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени  $t_2 < t_1$ , также выдавая информацию 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.

Создаем файл , а также открываем emacs в фоновом режиме. Делаем файл исполняемым (рис. 4.1)

```
[adchvanova@fedora ~]$ touch pr1.sh
\[adchvanova@fedora ~]$ emacs &
[1] 62554
[adchvanova@fedora ~]$ chm
chmem  chmod
[adchvanova@fedora ~]$ chmod +x pr1.sh
```

Рис. 4.1: Создание файла, открытие emacs в фоновом режиме. Команда `chmod +x`, которая дает право на исполнение.

Пишем программу.(рис. 4.2 - 4.3)



```
#!/bin/bash
function ozid
{
c1=$(date + "%s")
c2=$(date + "%s")
((t=c2-c1))
while ((t<t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date + "%s")
    ((t=s2-c1))
done
}
function vipoln
{
c1=$(date + "%s")
c2=$(date + "%s")
((t=c2-c1))
while ((t<t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date + "%s")
    ((t=s2-c1))
done
}
```

Рис. 4.2: Скрипт программы

```

t1=$1
t2=$2
com=$3
while true
do
    if [ "$com" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi

    if [ "$com" == "Ожидание" ]
    then
        echo ozid
    fi
    if [ "$com" == "Выполнение" ]
    then
        echo vipoln
    fi
    echo "Введите следующее действие"
    read com
done

```

Рис. 4.3: Скрипт программы

Проверяем его работу. (рис. 4.4)

```

[adchvanova@fedora ~]$ ./pr1.sh 2 3 Ожидание /home/adchvanova
ozid
Введите следующее действие
Выполнение
vipoln
Введите следующее действие
Выход
Выход
[adchvanova@fedora ~]$

```

Рис. 4.4: работа программы

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

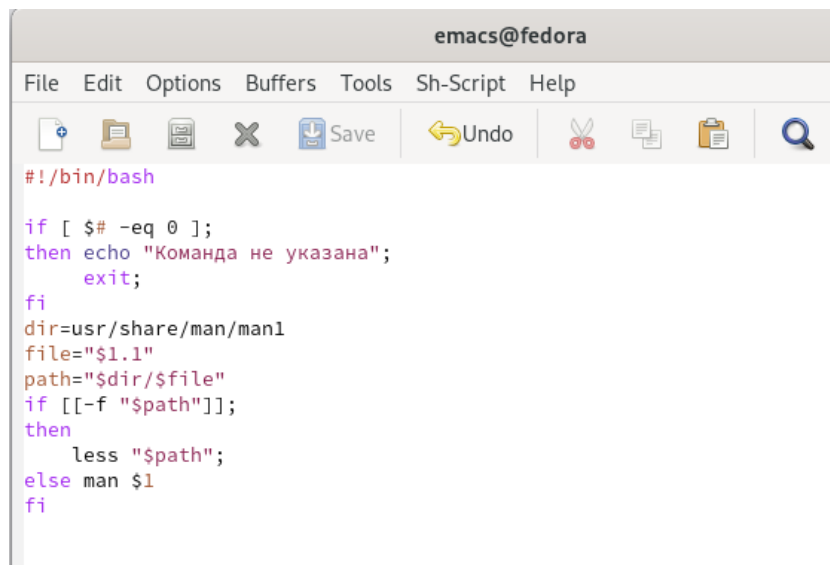
выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

Создаем файл , а также открываем emacs в фоновом режиме. Делаем файл исполняемым (рис. 4.5)

```
[adchvanova@fedora ~]$ ./pr1.sh 2 3 Ожидание /home/adchvanova
ozid
Введите следующее действие
Выполнение
vicoln
Введите следующее действие
Выход
Выход
[adchvanova@fedora ~]$
```

Рис. 4.5: Создание файла, открытие emacs в фоновом режиме. Команда chmod +x, которая дает право на исполнение.

Пишем программу.(рис. 4.6 )



```
emacs@fedora
File Edit Options Buffers Tools Sh-Script Help
+ Save Undo
#!/bin/bash

if [ $# -eq 0 ];
then echo "Команда не указана";
    exit;
fi
dir=usr/share/man/man1
file="$1.1"
path="$dir/$file"
if [[ -f "$path" ]];
then
    less "$path";
else man $1
fi
```

Рис. 4.6: Скрипт программы

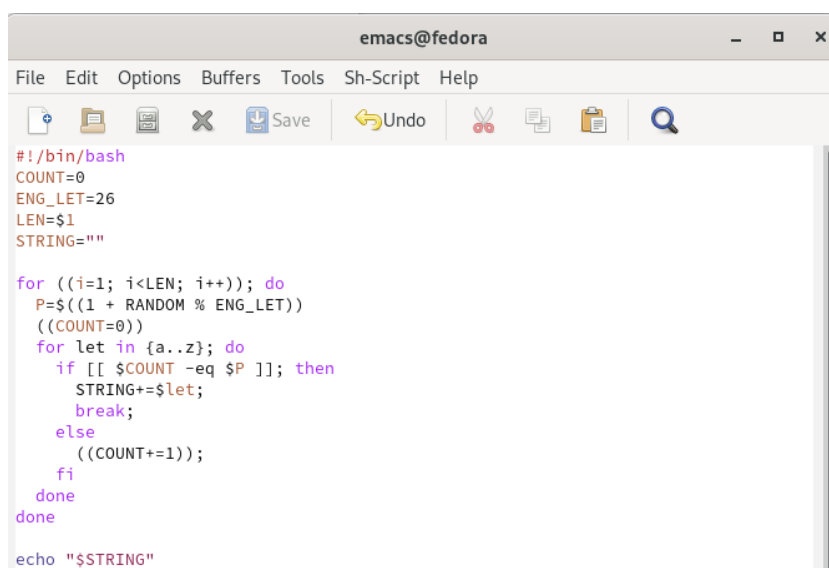
- Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Создаем файл , а также открываем emacs в фоновом режиме. Делаем файл исполняемым (рис. 4.7)

```
[adchvanova@fedora ~]$ touch pr3.sh
[adchvanova@fedora ~]$ emacs &
[1] 64379
[adchvanova@fedora ~]$ chmod +x pr3.sh
```

Рис. 4.7: Создание файла, открытие emacs в фоновом режиме. Команда chmod +x, которая дает право на исполнение.

Пишем программу.(рис. 4.8 )

The image shows a screenshot of the Emacs editor window titled 'emacs@fedora'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu bar is a toolbar with icons for file operations and editing. The main text area contains a shell script in bash. The script initializes variables: COUNT=0, ENG\_LET=26, LEN=\$1, and STRING="". It then enters a loop 'for ((i=1; i<LEN; i++)); do'. Inside the loop, it calculates a random index 'P=\$((1 + RANDOM % ENG\_LET))', resets COUNT to 0, and enters another loop 'for let in {a..z}; do'. This inner loop checks if the current character 'let' matches the random index 'P'. If it does, it appends the character to STRING and breaks the loop. If not, it increments COUNT. The outer loop ends with 'done', and the script concludes with 'echo "\$STRING"'.

```
#!/bin/bash
COUNT=0
ENG_LET=26
LEN=$1
STRING=""

for ((i=1; i<LEN; i++)); do
    P=$((1 + RANDOM % ENG_LET))
    ((COUNT=0))
    for let in {a..z}; do
        if [[ $COUNT -eq $P ]]; then
            STRING+=$let;
            break;
        else
            ((COUNT+=1));
        fi
    done
done
echo "$STRING"
```

Рис. 4.8: Скрипт программы

Проверяем его работу. (рис. 4.9)

```
[adchvanova@fedora ~]$ ./pr3.sh 8
dbeldep
[adchvanova@fedora ~]$ ./pr3.sh 9
msqkyvkł
[adchvanova@fedora ~]$ ./pr3.sh 9
orwczqvs
[adchvanova@fedora ~]$ ./pr3.sh 12
xuqgdvwcbww
[adchvanova@fedora ~]$ ./pr3.sh 5
qhtr
```

Рис. 4.9: работа программы

## 5 Выводы

Мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 6 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 cdc 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.