

## Лабораторная работа № 12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

---

Сильвен Макс Грегор Филс , НКАбд-03-22

27 Апрель 2023

Российский университет дружбы народов, Москва, Россия

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

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.

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 оболочкой 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

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 (**Prog:bash?**).

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 имела возможность взаимодействия трёх и более процессов. (рис. (fig:001?; fig:002?))

# Выполнение лабораторной работы



```
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 unblocked"
13    flock -u ${fn}
14 else
15    echo "File is blocked"
16    sleep 5
17    fi
18 done
```

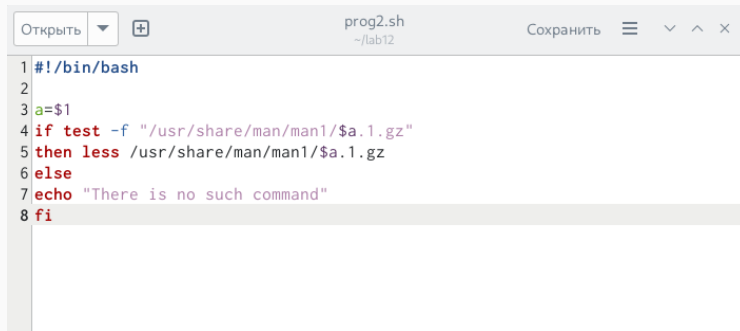
Рис. 1: Текст первой программы



## Выполнение лабораторной работы

[illegible]

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`. (рис. (fig:003?; fig:004?; fig:005?))

A screenshot of a code editor window. The title bar shows the filename 'prog2.sh' and the path '~/lab12'. On the left, there is a button labeled 'Открыть' (Open) and a dropdown menu. On the right, there is a button labeled 'Сохранить' (Save) and three icons: a hamburger menu, a downward arrow, and an upward arrow. The editor area contains a shell script with 8 lines of code, each preceded by a line number. The code is as follows:

```
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
```

Рис. 3: Текст второй программы

# Выполнение лабораторной работы

```
Терминал - fsmaksgregor@dk2n25:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk2n25:~/lab12  fsmaksgregor@dk2n25:~

MKDIR(1)                                     User Commands                                     MKDIR(1)

NAME
    mkdir - make directories

SYNOPSIS
    mkdir [OPTION]... DIRECTORY...

DESCRIPTION
    Create the DIRECTORY(ies), if they do not already exist.

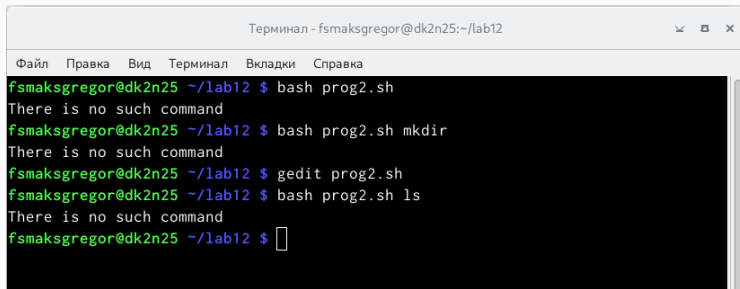
    Mandatory arguments to long options are mandatory for short options
    too.

    -m, --mode=MODE
        set file mode (as in chmod), not a=rwx - umask

    -p, --parents
        no error if existing, make parent directories as needed, with
        their file modes unaffected by any -m option.

    -v, --verbose
        print a message for each created directory

Manual page mkdir(1) line 1 (press h for help or q to quit)
```

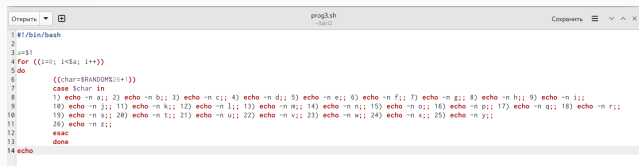


The image shows a terminal window titled "Терминал - fsmaksgregor@dk2n25:~/lab12". The window has a menu bar with "Файл", "Правка", "Вид", "Терминал", "Вкладки", and "Справка". The terminal content shows the user fsmaksgregor@dk2n25 in the directory ~/lab12. They enter the command `bash prog2.sh`, which results in the message "There is no such command". They then enter `bash prog2.sh mkdir`, also resulting in "There is no such command". Next, they enter `gedit prog2.sh`, and finally `bash prog2.sh ls`, which again results in "There is no such command". The terminal ends with the prompt `fsmaksgregor@dk2n25 ~/lab12 $` and a cursor.

```
Терминал - fsmaksgregor@dk2n25:~/lab12
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk2n25 ~/lab12 $ bash prog2.sh
There is no such command
fsmaksgregor@dk2n25 ~/lab12 $ bash prog2.sh mkdir
There is no such command
fsmaksgregor@dk2n25 ~/lab12 $ gedit prog2.sh
fsmaksgregor@dk2n25 ~/lab12 $ bash prog2.sh ls
There is no such command
fsmaksgregor@dk2n25 ~/lab12 $
```

Рис. 5: Результат

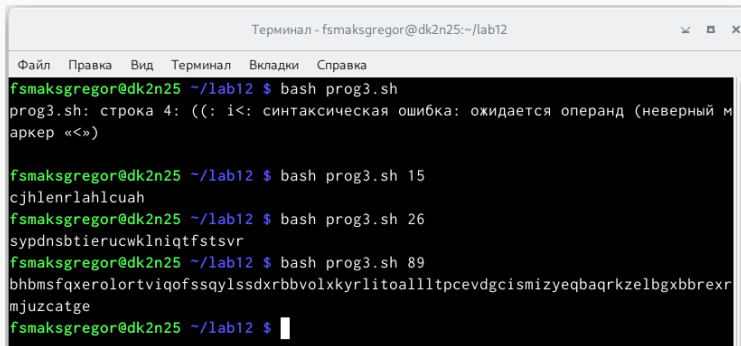
3. Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. (рис. (fig:006?; fig:007?))



```
1 #!/bin/bash
2
3 a=51
4 for ((i=0; i<$a; i++))
5 do
6     ((char=$RANDOM%26+1))
7     case $char in
8         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;;
9         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;;
10        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;;
11        26) echo -n z;;
12    esac
13    done
14 echo
```

Рис. 6: Текст третьей программы

# Выполнение лабораторной работы



```
Терминал - fsmaksgregor@dk2n25:~/lab12
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk2n25 ~/lab12 $ bash prog3.sh
prog3.sh: строка 4: ((: i<: синтаксическая ошибка: ожидается операнд (неверный м
аркер «<»)

fsmaksgregor@dk2n25 ~/lab12 $ bash prog3.sh 15
cjhlenrlahlcuah
fsmaksgregor@dk2n25 ~/lab12 $ bash prog3.sh 26
sypdnsbtierucwklniqtfstsvr
fsmaksgregor@dk2n25 ~/lab12 $ bash prog3.sh 89
bhbmsfqxerolortviqofssqylssdxrbbvolxkyrlitoalltpcevdgcismizyeqbaqrkzelbgxbbrex
mjuzcatge
fsmaksgregor@dk2n25 ~/lab12 $
```

Рис. 7: Результат

- В процессе выполнения этой лабораторной работы я продолжил осваивать программирование на `bash`.



Спасибо за внимание!