

Отчёт по лабораторной работе №12

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

Надежда Александровна Рогожина

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Выводы	15
	Список литературы	16

Список иллюстраций

4.1	Скрипт №1	11
4.2	Выполнение скрипта №1	11
4.3	Реализация команды map	12
4.4	Пример использования	13
4.5	Скрипт №3	13
4.6	Примеры выполнения	14

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

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 Foundation).

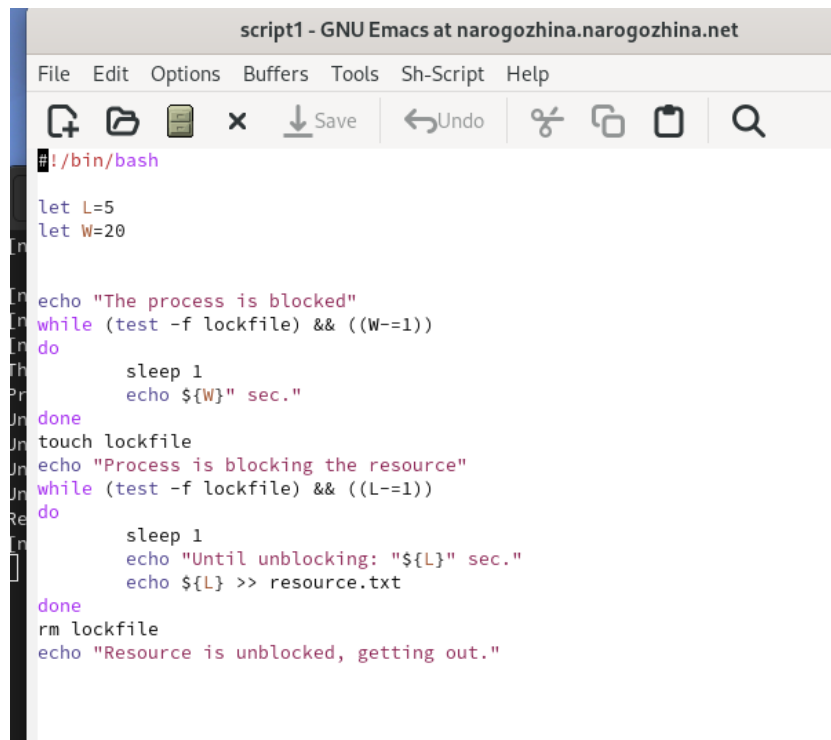
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

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 Керна.

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 (рис. 4.1, 4.2):



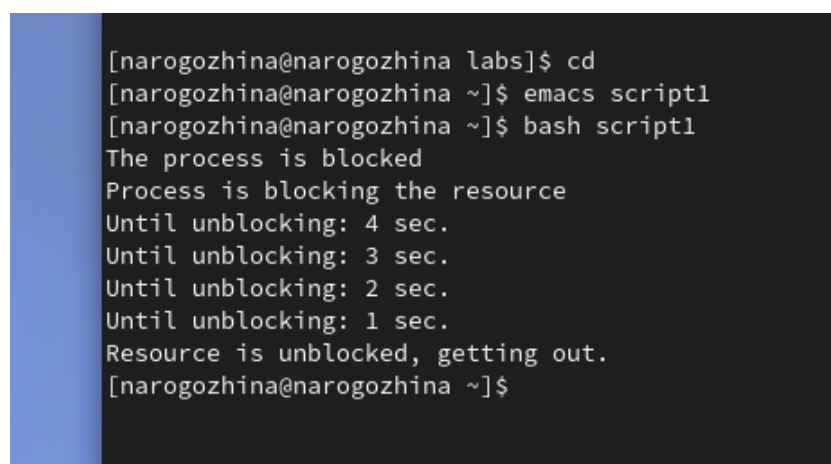
```
script1 - GNU Emacs at narogozhina.narogozhina.net
File Edit Options Buffers Tools Sh-Script Help
[Icons: Copy, Paste, Save, Undo, Cut, Copy, Paste, Search]

#!/bin/bash

let L=5
let W=20

echo "The process is blocked"
while (test -f lockfile) && ((W-=1))
do
    sleep 1
    echo "${W}" sec."
done
touch lockfile
echo "Process is blocking the resource"
while (test -f lockfile) && ((L-=1))
do
    sleep 1
    echo "Until unblocking: "${L}" sec."
    echo ${L} >> resource.txt
done
rm lockfile
echo "Resource is unblocked, getting out."
```

Рис. 4.1: Скрипт №1



```
[narogozhina@narogozhina labs]$ cd
[narogozhina@narogozhina ~]$ emacs script1
[narogozhina@narogozhina ~]$ bash script1
The process is blocked
Process is blocking the resource
Until unblocking: 4 sec.
Until unblocking: 3 sec.
Until unblocking: 2 sec.
Until unblocking: 1 sec.
Resource is unblocked, getting out.
[narogozhina@narogozhina ~]$
```

Рис. 4.2: Выполнение скрипта №1

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. 4.3,4.4):

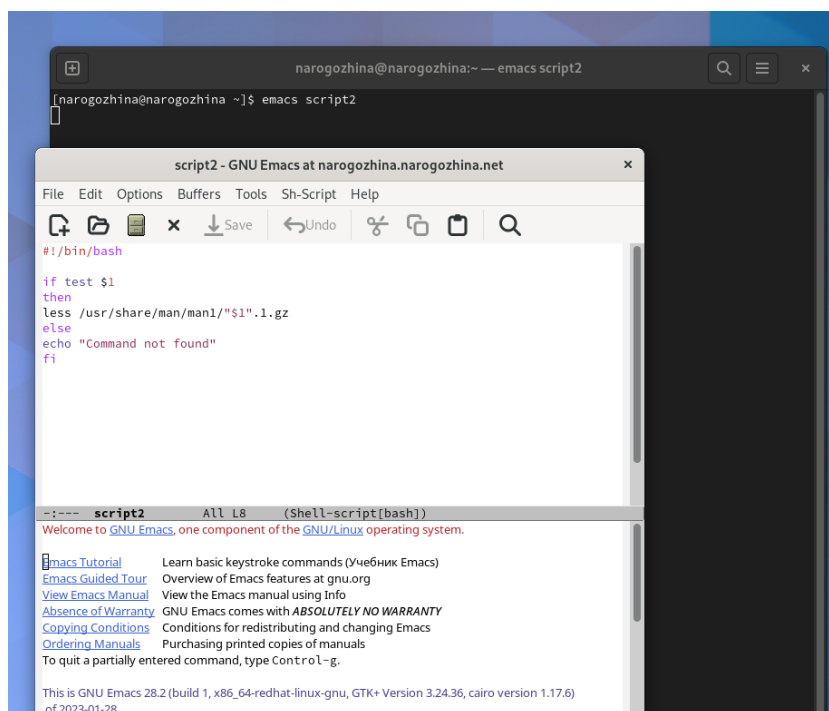
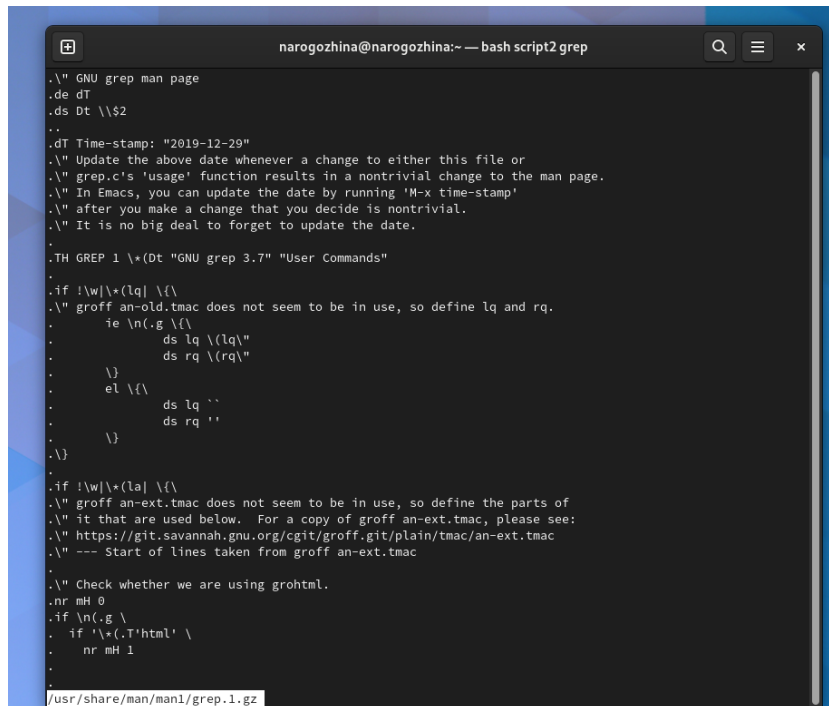


Рис. 4.3: Реализация команды man



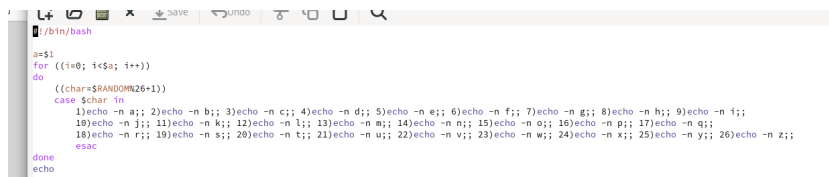
```
narogozhina@narogozhina:~ — bash script2 grep
.\" GNU grep man page
.de dT
.ds Dt \\$2
..
.dT Time-stamp: "2019-12-29"
.\" Update the above date whenever a change to either this file or
.\" grep.c's 'usage' function results in a nontrivial change to the man page.
.\" In Emacs, you can update the date by running 'M-x time-stamp'
.\" after you make a change that you decide is nontrivial.
.\" It is no big deal to forget to update the date.

.TH GREP 1 \*(Dt "GNU grep 3.7" "User Commands"
.
.if !\w|\*(lq| \{\
.\" groff an-old.tmac does not seem to be in use, so define lq and rq.
.   ie \n(.g \{\
.     ds lq \(\lq\"
.     ds rq \(\rq\"
.   \}
.   el \{\
.     ds lq ``
.     ds rq ''
.   \}
.\}

.if !\w|\*(la| \{\
.\" groff an-ext.tmac does not seem to be in use, so define the parts of
.\" it that are used below.  For a copy of groff an-ext.tmac, please see:
.\" https://git.savannah.gnu.org/cgiit/groff.git/plain/tmac/an-ext.tmac
.\" --- Start of lines taken from groff an-ext.tmac
.
.\" Check whether we are using grohtml.
.nr mH 0
.if \n(.g \
.  if '\*(.T'html' \
.    nr mH 1
.
.usr/share/man/man1/grep.1.gz
```

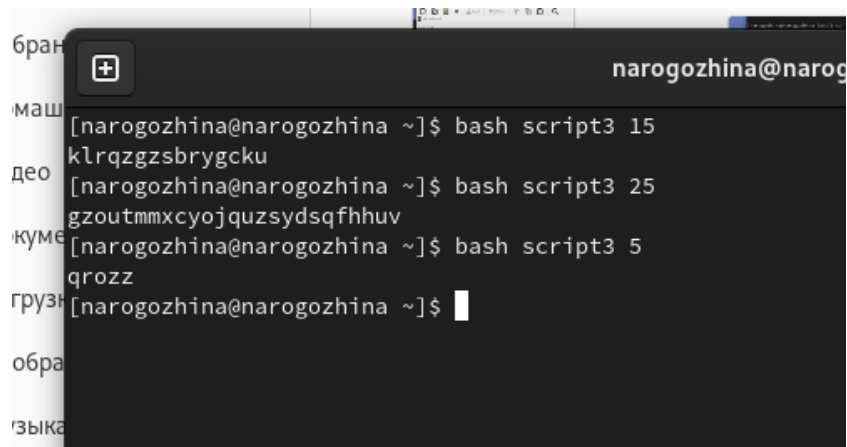
Рис. 4.4: Пример использования

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. 4.5,4.6):



```
narogozhina@narogozhina:~ — bash script2 grep
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 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. 4.5: Скрипт №3

A terminal window titled 'narogozhina@narog' with a '+' icon in the top-left corner. The window shows a series of commands and their outputs. The first command is 'bash script3 15', which outputs 'klrqzgzsbyrgcku'. The second command is 'bash script3 25', which outputs 'gzoutmmxcyojqzsydsqfhuv'. The third command is 'bash script3 5', which outputs 'qrozz'. The prompt '[narogozhina@narogozhina ~]\$' is visible at the end of the last line.

```
[narogozhina@narogozhina ~]$ bash script3 15
klrqzgzsbyrgcku
[narogozhina@narogozhina ~]$ bash script3 25
gzoutmmxcyojqzsydsqfhuv
[narogozhina@narogozhina ~]$ bash script3 5
qrozz
[narogozhina@narogozhina ~]$
```

Рис. 4.6: Примеры выполнения

5 Выводы

В ходе лабораторной работы мы научились писать командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы

1. Руководство по выполнению лабораторной работы №10
2. Руководство по выполнению лабораторной работы №12