Лабораторная работа №12

Отчёт по лабораторной работе №12

Макарова Анастасия Михайловна

Содержание

Цель работы

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

Выполнение лабораторной работы

1. Откроем редактор emacs с помощью команды emacs & и создадим в нем файл semafor.sh (Рис.1).

Рис.1

Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом) (Рис.2).

```
#!/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:**-  semafor.sh      Top L24     (Shell-script[sh])
```

Рис.2

Передадим нашему файлу права на выполнение с помощью команды chmod с опцией +x, затем проверим работу файла командой ./semafor.sh 3 7 (Рис.3).

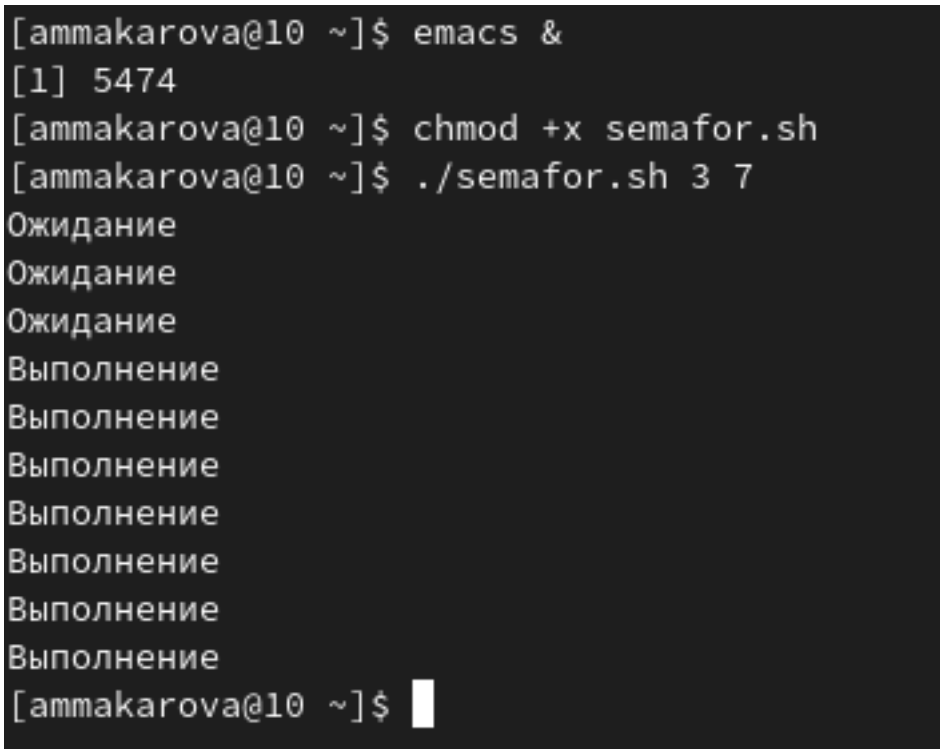

Рис.3

Изменим программу таким образом, чтобы можно было 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.4, 5).

```
#!/bin/bash
function ozgidanie
{
    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
U:**-  semafor.sh     Top L28    (Shell-script[sh])
```

Рис.4

```
    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
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ozgidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done
```

```
U:**-  semafor.sh     43% L36    (Shell-script[sh])
```

Рис.5

Проверим работу файла с помощью команды ./semafor.sh 2 4 Ожидание > /dev/pts/1 &. Хотя нам и отказано в доступе, программа работает верно (Рис.6, 7).

```
[ammakarova@10 ~]$ ./semafor.sh 2 4 Ожидание > /dev/pts/1 &
[2] 5839
bash: /dev/pts/1: Отказано в доступе
[2]+  Выход 1            ./semafor.sh 2 4 Ожидание > /dev/pts/1
[ammakarova@10 ~]$ ./semafor.sh 2 4 Выполнение > /dev/pts/2 &
[2] 5853
bash: /dev/pts/2: Отказано в доступе
[2]+  Выход 1            ./semafor.sh 2 4 Выполнение > /dev/pts/2
```

Рис.6

```
[ammakarova@10 ~]$ ./semafor.sh 2 4 Ожидание
Ожидание
Ожидание
Следующее действие:
Ожидание
Ожидание
Ожидание
Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:

Следующее действие:
Выход
Выход
[ammakarova@10 ~]$
```

Рис.7

2. Изучим содержимое каталога /usr/share/man/man1 с помощью команды ls. 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 (Рис.8, 9).

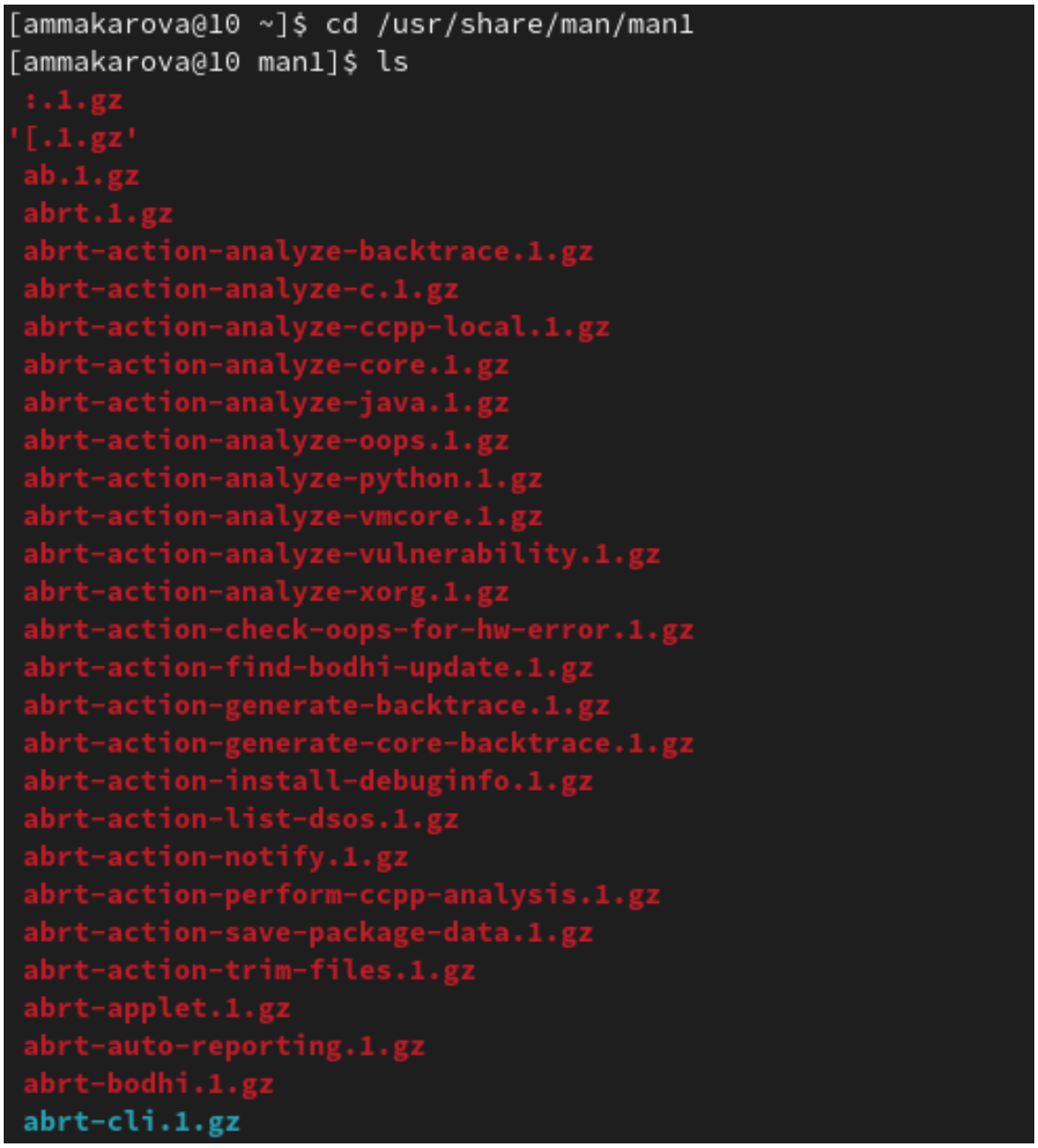
[ammakarova@10 ~]$ cd /usr/share/man/man1
[ammakarova@10 man1]$ ls
:.1.gz
'[.1.gz'
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz

Рис.8

Рис.9

Реализуем команду man с помощью командного файла. Создаем в редакторе emacs файл man.sh (Рис.10). 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.11).

```
Find file: ~/man.sh
```

Рис.10

```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.gz ]
then
    gunzip -c /usr/share/man/man1/$1.1.gz | less
else
    echo "Нет справки по данной команде"
fi
```

```
U:---  man.sh          All L1     (Shell-script[bash])
```

Рис.11

Добавим права на выполнение программы с помощью команды chmod с опцией +x и проверим работу файла (Рис.12).

```
[ammakarova@10 ~]$ chmod +x man.sh
[ammakarova@10 ~]$ ./man.sh mkdir
[ammakarova@10 ~]$ ./man.sh cat
[ammakarova@10 ~]$ ./man.sh rm
[ammakarova@10 ~]$ ./man.sh ls
```

Рис.12

Результаты работы программ (Рис.13-16):

```
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.47.3.
.TH MKDIR "1" "March 2022" "GNU coreutils 8.32" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fI\,OPTION\/\fR]... \fI\,DIRECTORY\/\fR...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Create the DIRECTORY(ies), if they do not already exist.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\-m\fR, \fB\-\-mode\fR=\fI\,MODE\/\fR
set file mode (as in chmod), not a=rwx \- umask
.TP
\fB\-p\fR, \fB\-\-parents\fR
no error if existing, make parent directories as needed
.TP
\fB\-v\fR, \fB\-\-verbose\fR
print a message for each created directory
.TP
\fB\-Z\fR
set SELinux security context of each created directory
to the default type
.TP
\fB\-\-context\fR[=\fI\,CTX\/\fR]
like \fB\-Z\fR, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
:
```

Рис.13

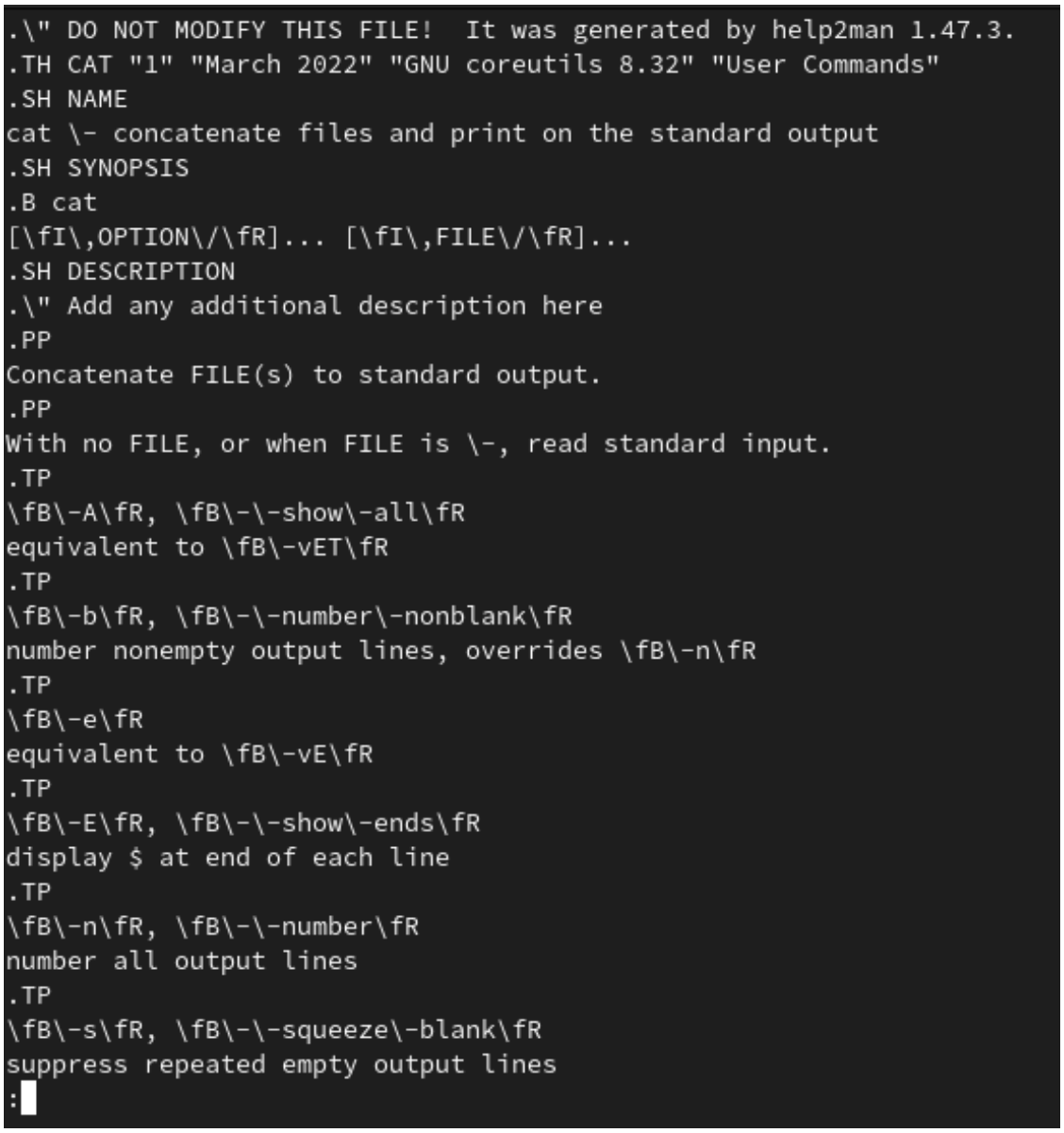
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH CAT "1" "March 2022" "GNU coreutils 8.32" "User Commands"
.SH NAME
cat \- concatenate files and print on the standard output
.SH SYNOPSIS
.B cat
[\fI\,OPTION\/\fR]... [\fI\,FILE\/\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Concatenate FILE(s) to standard output.
.PP
With no FILE, or when FILE is \-, read standard input.
.TP
\fB\-A\fR, \fB\-\-show\-all\fR
equivalent to \fB\-vET\fR
.TP
\fB\-b\fR, \fB\-\-number\-nonblank\fR
number nonempty output lines, overrides \fB\-n\fR
.TP
\fB\-e\fR
equivalent to \fB\-vE\fR
.TP
\fB\-E\fR, \fB\-\-show\-ends\fR
display $ at end of each line
.TP
\fB\-n\fR, \fB\-\-number\fR
number all output lines
.TP
\fB\-s\fR, \fB\-\-squeeze\-blank\fR
suppress repeated empty output lines
:

Рис.14

```
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.47.3.
.TH RM "1" "March 2022" "GNU coreutils 8.32" "User Commands"
.SH NAME
rm \- remove files or directories
.SH SYNOPSIS
.B rm
[\fI\,OPTION\/\fR]... [\fI\,FILE\/\fR]...
.SH DESCRIPTION
This manual page
documents the GNU version of
.BR rm .
.B rm
removes each specified file.  By default, it does not remove
directories.
.P
If the \fI\-I\fR or \fI\-\-interactive=once\fR option is given,
and there are more than three files or the \fI\-r\fR, \fI\-R\fR,
or \fI\-\-recursive\fR are given, then
.B rm
prompts the user for whether to proceed with the entire operation.  If
the response is not affirmative, the entire command is aborted.
.P
Otherwise, if a file is unwritable, standard input is a terminal, and
the \fI\-f\fR or \fI\-\-force\fR option is not given, or the
\fI\-i\fR or \fI\-\-interactive=always\fR option is given,
.B rm
prompts the user for whether to remove the file.  If the response is
not affirmative, the file is skipped.
.SH OPTIONS
.PP
Remove (unlink) the FILE(s).
:
```

Puc.15

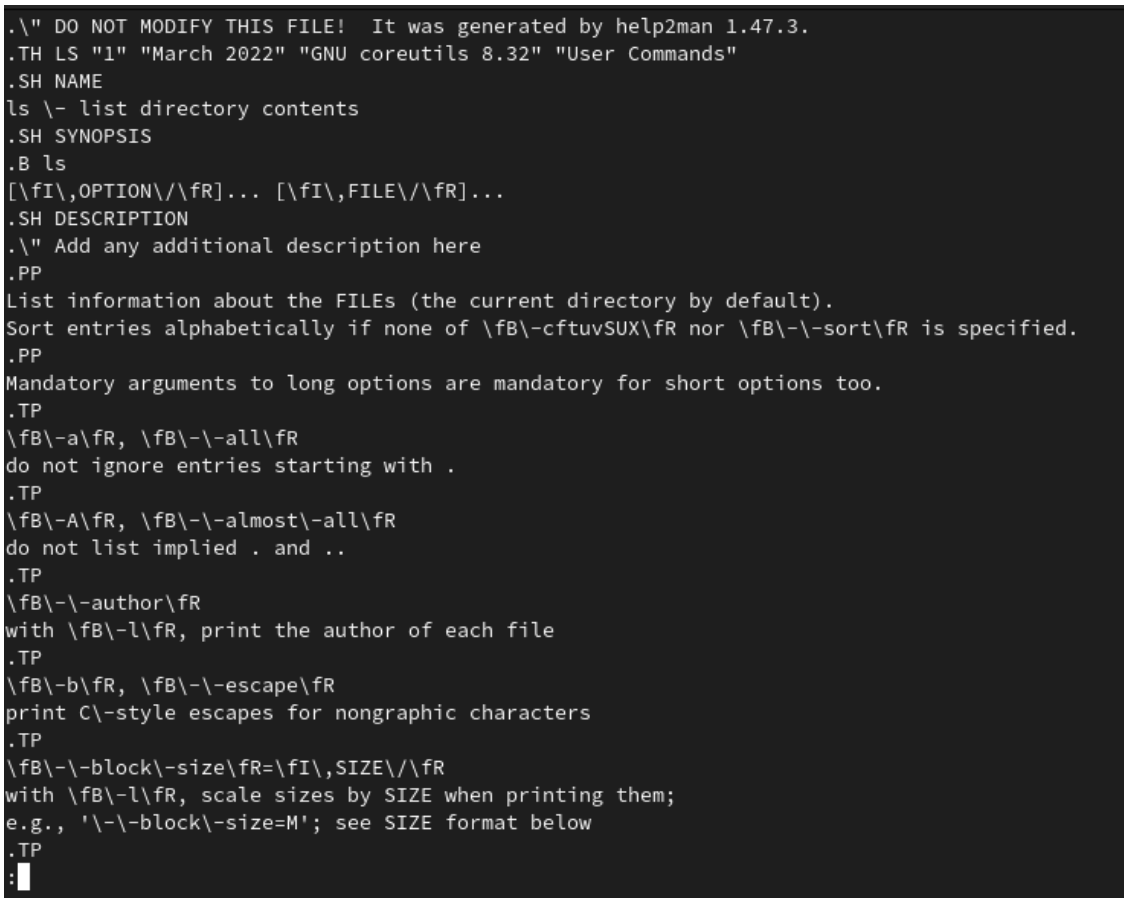

Рис.16

3. Создадим в редакторе emacs файл random.sh (Рис.17). Используя встроенную переменную $RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. Учитываем, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767 (Рис.18, 19).

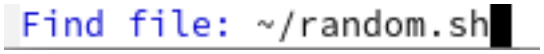

Рис.17

```
#!/bin/bash
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;;
U:---  random.sh      Top L28    (Shell-script[sh])
```

Рис.18

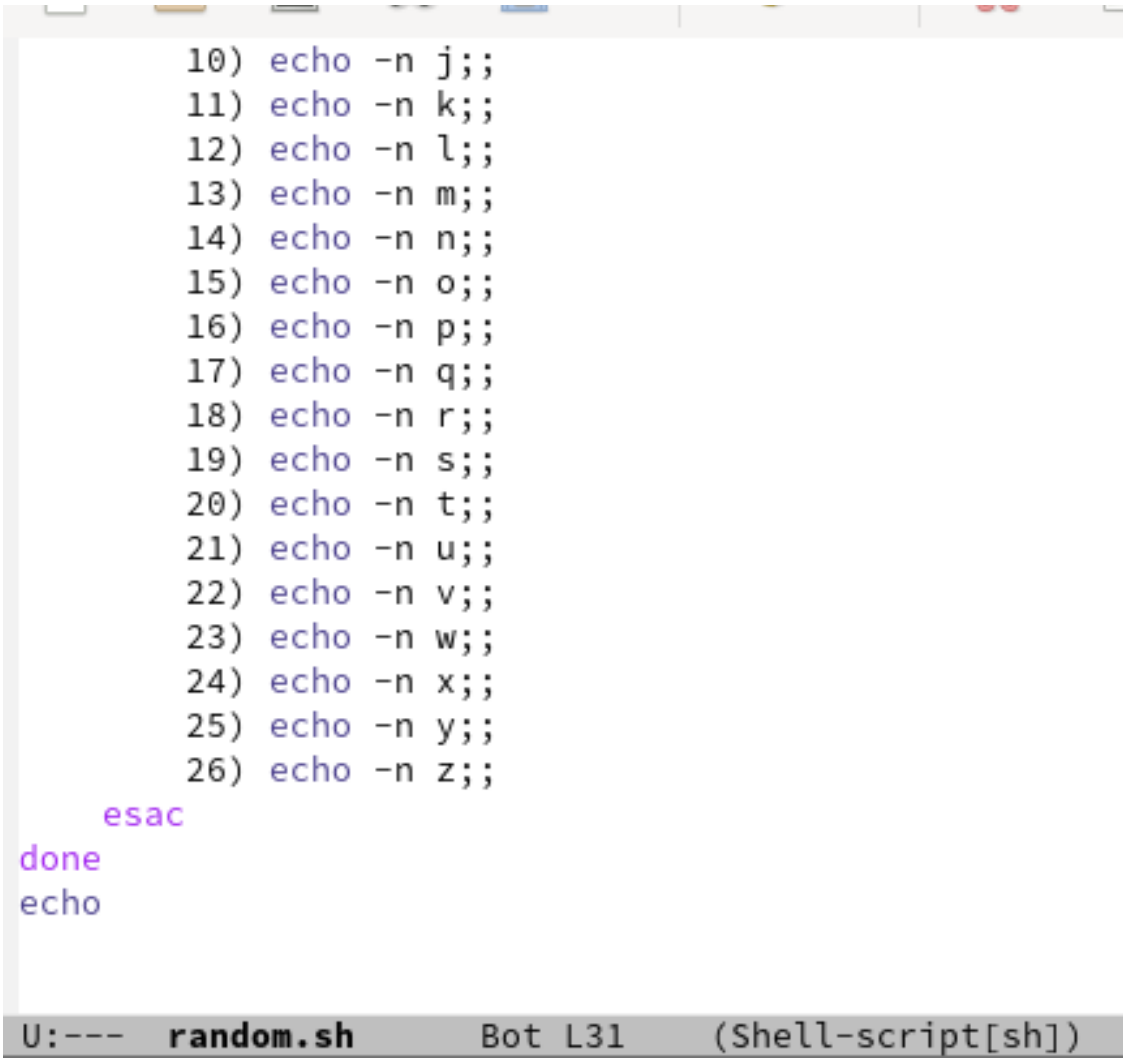

Рис.19

Добавим права на выполнение программы с помощью команды chmod с опцией +x и проверим работу файла (Рис.20).

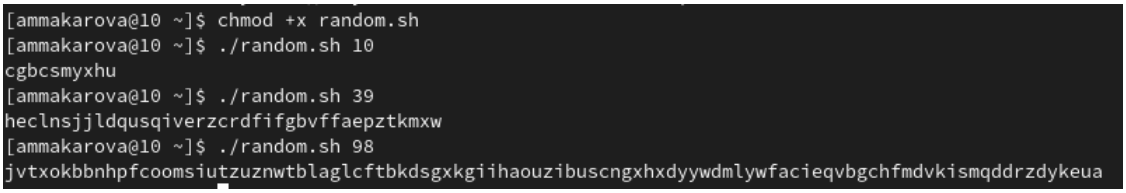

Рис.20

Вывод

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать более

сложные командные файлы с использованием логических управляющих конструкций и циклов.

Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

- while [$1 != “exit”]

1) Не хватает пробела после первой скобки и перед второй скобкой;
2) $1 необходимо взять в кавычки, потому что эта переменная может содержать пробелы.

2. Как объединить (конкатенация) несколько строк в одну? Самый простой способ объединить две или более строковых переменных — записать их одну за другой. Вы также можете объединить одну или несколько переменных с помощью буквальных строк: VAR1=“Hello,” VAR2=“${VAR1}World" echo "$VAR2”

Hello, World

В приведенном выше примере переменная VAR1 заключена в фигурные скобки, чтобы защитить имя переменной от окружающих символов. Если за переменной следует другой допустимый символ имени переменной, вы должны заключить его в фигурные скобки ${VAR1} . Чтобы избежать проблем с разделением слов или подстановкой слов, вы всегда должны стараться заключать имя переменной в двойные кавычки. Если вы хотите подавить интерполяцию переменных и особую обработку символа обратной косой черты вместо двойных, используйте одинарные кавычки. Bash не разделяет переменные по «типу», переменные обрабатываются как целые или строковые в зависимости от контекстов. Вы также можете объединять переменные, содержащие только цифры. Другой способ объединения строк в bash — это добавление переменных или буквальных строк к переменной с помощью оператора +=

3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash? Используется для генерации чисел от первого до последнего шага increment. Если вы запускаете seq с одним числом в качестве параметра командной строки, он считается от единицы до этого числа. Затем он печатает числа в окне терминала, по одному числу в строке. Мы также можем попросить seq создать список чисел от самого высокого до самого низкого.

4. Какой результат даст вычисление выражения $((10/3))? Результат будет 3, т.к. это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки zsh от bash. Подобно Bash, Z Shell можно в общем рассматривать как расширенную версию Bourne Shell, и она содержит много черт, сходных с Bash, что вы

сможете заметить ниже. Также можно видеть, что она довольно сильно напоминает Korn Shell. Вот список того, что вы получите, используя Z Shell вместо Bash:

- Встроенная команда zmv поможет вам массово переименовать файлы/директории, например, чтобы добавить '.txt' к имени каждого файла, запустите zmv –C '(*)(#q.)' '$1.txt'.
- Утилита zcalc — это замечательный калькулятор командной строки, это удобный способ считать быстро, не покидая терминал. Загрузите её через autoload -Uz zcalc и запустите командой zcalc.
- Команда zparseopts — это однострочник, который поможет вам разобрать сложные варианты, которые предоставляются вашему скрипту(?)
- Команда autopushd позволяет вам делать popd после того, как вы с помощью cd, чтобы вернуться в предыдущую директорию.
- Поддержка чисел с плавающей точкой (коей Bash, к удивлению, не содержит).
- Поддержка для структур данных "хэш".

6. Проверьте, верен ли синтаксис данной конструкции:

- for ((a=1; a <= LIMIT; a++)) Конуструкция верна.

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки? Bash — это командный язык, а не язык программирования общего назначения. Поэтому с усложнением логики вашего автоматизированного сценария он становится более запутанным и менее читаемым. Кроме того, Bash все и всегда воспринимает как команду, потому что это командный язык. Если вам нужно часто инициировать процессы и писать небольшой переносимый Shell-сценарий для Unix или Unix-подобных операционных систем, Bash, несомненно, будет хорошим выбором. Если вам нужно написать кроссплатформенный Shell-скрипт для обработки некоторых данных и выполнения определенных команд, можете выбрать Python. JavaScript отлично подходит для тех же сценариев, что и Python. Однако, в отличие от Python, JavaScript имеет некоторые дополнительные преимущества. JavaScript быстр, изначально поддерживает JSON и имеет впечатляющие встроенные функции.