

Отчет по лабораторной работе 12

Лабораторная работа 12

Куркина Евгения Вячеславовна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	18
	Список литературы	19

Список иллюстраций

3.1	Создание файла, переход в emacs	7
3.2	Текст скрипта	8
3.3	Права на выполнение и результат работы файла	9
3.4	Измененный текст скрипта ч1	10
3.5	Измененный текст скрипта ч2	10
3.6	Проверка командного файла	11
3.7	Путь в каталог	11
3.8	Список файлов	11
3.9	Создание нового файла и переход в emacs	12
3.10	Текст командного файла	12
3.11	Права доступа, команды проверки файла	12
3.12	Справка по команде	13
3.13	Создание нового файла для скрипта 3	13
3.14	Текст скрипта3	14
3.15	Права доступа результат проверки исполнения файла	14

Список таблиц

1 Цель работы

Здесь приводится формулировка цели лабораторной работы. Формулировки цели для каждой лабораторной работы приведены в методических указаниях.

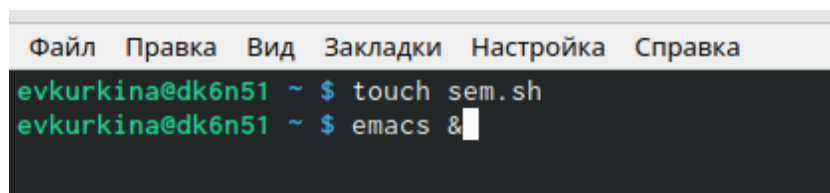
Цель данной лабораторной работы — Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

Написать необходимые командные файлы.

3 Выполнение лабораторной работы

- 1) Создала отдельный файл sem.sh(рис. 3.1). Написала командный файл, который реализует упрощенный механизм семафоров. Команда должна дожидаться освобождения ресурса, выдавая об этом сообщение, а затем использовать его в течении некоторого времени, также выдавая информацию об этом (рис. 3.2). Затем дала права на выполнение, а затем проверила результат работы файла (рис. 3.3).



```
Файл  Правка  Вид  Закладки  Настройка  Справка
evkurkina@dk6n51 ~ $ touch sem.sh
evkurkina@dk6n51 ~ $ emacs &
```

Рис. 3.1: Создание файла, переход в emacs



```
#!/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
```

Рис. 3.2: Текст скрипта


```
evkurkina@dk6n51 ~ $ chmod +x sem.sh
evkurkina@dk6n51 ~ $ ./sem.sh 4 7
Ожидание
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
evkurkina@dk6n51 ~ $
```

Рис. 3.3: Права на выполнение и результат работы файла

1.2) Изменила текст скрипта так, чтобы его можно было выполнять сразу в нескольких терминалах (рис. 3.4)(рис. 3.5), затем проверила его работу (нет прав доступа для данной команды) (рис. 3.6).

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
function ogidanie
{
    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
}
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
        echo "Выход"
        exit 0
    fi
    if [ "$command" "" "Ожидание" ]
    then ogidanie
```

Рис. 3.4: Измененный текст скрипта ч1

```
fi
if [ "$command" "" "Ожидание" ]
then ogidanie
fi
if [ "$command" "" "Выполнение" ]
then vipolnenie
fi
echo "Следующее действие: "
read command
done
```

Рис. 3.5: Измененный текст скрипта ч2

```
evkurkina@dk6n51 ~ $ ./sem.sh 2 3 Ожидание > /dev/pts/1 &
[2] 12528
evkurkina@dk6n51 ~ $ bash: /dev/pts/1: Отказано в доступе
```

Рис. 3.6: Проверка командного файла

- 2) Перешла в каталог man1 (рис. 3.7), и командой ls просмотрела содержимое (рис. 3.8). Далее создала файл man.sh (рис. 3.9). Написала текст скрипта, который получает в виде аргумента название команды, и выдавать справку об этой команде или сообщение о том, что справка отсутствует (рис. 3.10). Затем дала права на исполнение и проверила работу командного файла, получив справку о команде rm (рис. 3.11) (рис. 3.12).

```
evkurkina@dk6n51 ~ $ cd /usr/share/man/man1
```

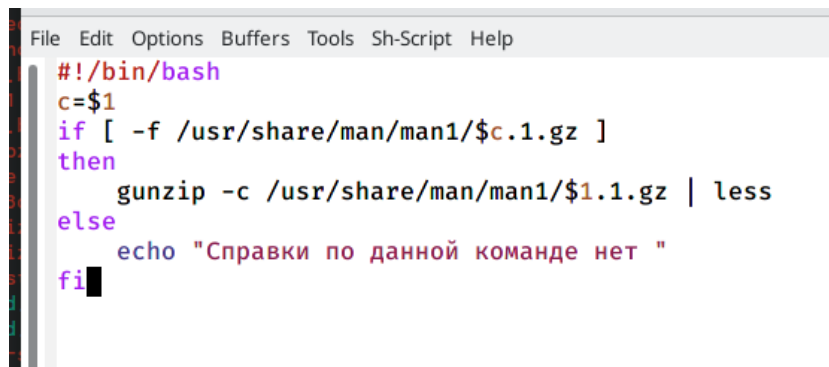
Рис. 3.7: Путь в каталог

```
notepad.1
notepad.1-staging
notify-send.1.bz2
nping.1.bz2
npm.1.bz2
npm-access.1.bz2
npm-adduser.1.bz2
npm-audit.1.bz2
npm-bin.1.bz2
npm-bugs.1.bz2
npm-build.1.bz2
npm-bundle.1.bz2
npm-cache.1.bz2
npm-ci.1.bz2
npm-completion.1.bz2
npm-config.1.bz2
npm-dedupe.1.bz2
npm-deprecate.1.bz2
npm-dist-tag.1.bz2
npm-docs.1.bz2
npm-doctor.1.bz2
npm-edit.1.bz2
npm-explore.1.bz2
npm-fund.1.bz2
npm-help.1.bz2
npm-help-search.1.bz2
npm-hook.1.bz2
npm-init.1.bz2
npm-install.1.bz2
npm-install-ci-test.1.bz2
npm-install-test.1.bz2
npm-link.1.bz2
npm-logout.1.bz2
npm-ls.1.bz2
npm-org.1.bz2
npm-outdated.1.bz2
npm-owner.1.bz2
zipnote.1.bz2
ziptool.1.bz2
zless.1.bz2
zlib_decompress.1.bz2
zlib-flate.1.bz2
zmore.1.bz2
znew.1.bz2
zonetab2pot.py.1.bz2
zresample.1.bz2
zretune.1.bz2
zrun.1.bz2
zsh.1.bz2
zshall.1.bz2
zshbuiltins.1.bz2
zshcalsys.1.bz2
zshcompctl.1.bz2
zshcompsys.1.bz2
zshcompwid.1.bz2
zshcontrib.1.bz2
zshexpn.1.bz2
zshmisc.1.bz2
zshmodules.1.bz2
zshoptions.1.bz2
zshparam.1.bz2
zshroadmap.1.bz2
zshrcpsys.1.bz2
zshzftpsys.1.bz2
zshzle.1.bz2
zsoelim.1.bz2
zstd.1.bz2
zstdcat.1.bz2
zstdgrep.1.bz2
zstdless.1.bz2
zvbi-chains.1.bz2
zvbid.1.bz2
zvbi-ntsc-cc.1.bz2
evkurkina@dk6n51 /usr/share/man/man1 $
```

Рис. 3.8: Список файлов

```
evkurkina@dk6n51 ~ $ touch man.sh
evkurkina@dk6n51 ~ $ emacs &
```

Рис. 3.9: Создание нового файла и переход в emacs

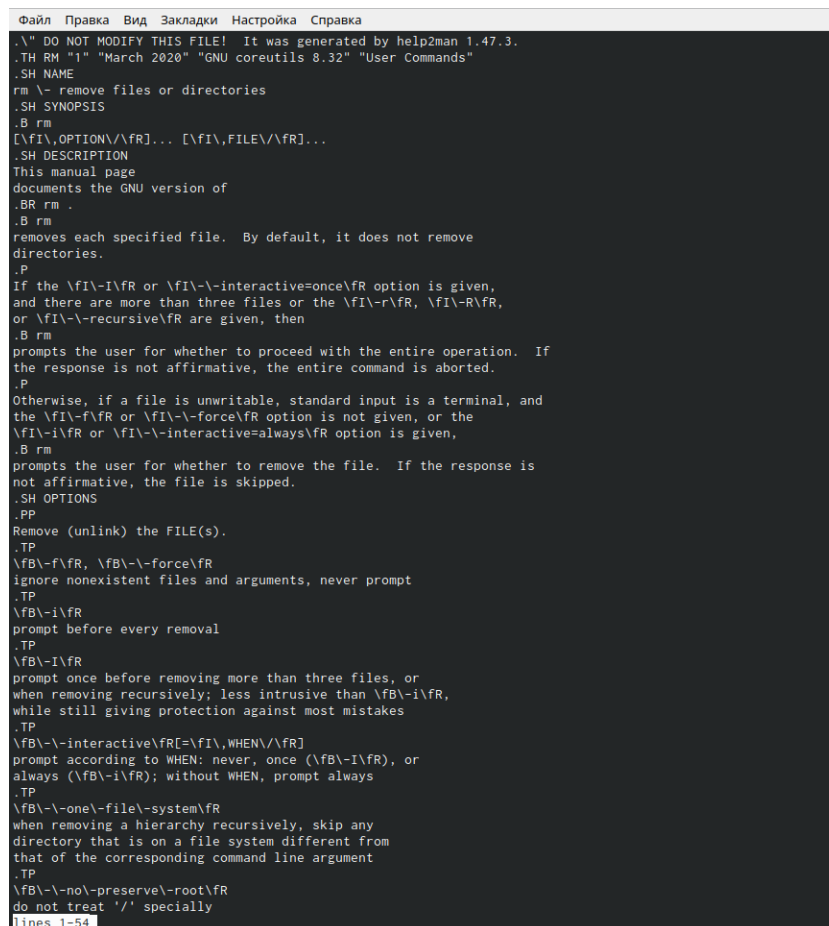
A screenshot of the Emacs text editor interface. The menu bar at the top includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The main text area contains a shell script with the following content:

```
#!/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. 3.10: Текст командного файла

```
evkurkina@dk6n51 ~ $ chmod +x man.sh
evkurkina@dk6n51 ~ $ ./man.sh rm
```

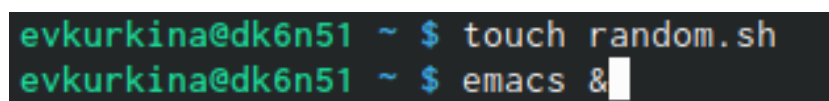
Рис. 3.11: Права доступа, команды проверки файла



```
Файл  Правка  Вид  Закладки  Настройка  Справка
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH RM "1" "March 2020" "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
.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).
.TP
\fB\--force\fR, \fB\--force\fR
ignore nonexistent files and arguments, never prompt
.TP
\fB\--i\fR
prompt before every removal
.TP
\fB\--I\fR
prompt once before removing more than three files, or
when removing recursively; less intrusive than \fB\--i\fR,
while still giving protection against most mistakes
.TP
\fB\--interactive\fR[=\fI\,WHEN\...\fR]
prompt according to WHEN: never, once (\fB\--I\fR), or
always (\fB\--i\fR); without WHEN, prompt always
.TP
\fB\--one-file-system\fR
when removing a hierarchy recursively, skip any
directory that is on a file system different from
that of the corresponding command line argument
.TP
\fB\--no-preserve-root\fR
do not treat '/' specially
lines 1-54
```

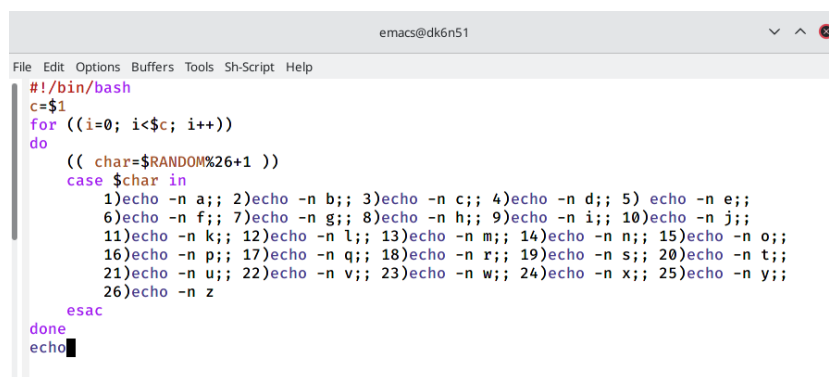
Рис. 3.12: Справка по команде

3) Создала новый файл с названием random.sh (рис. 3.13). Написала текст командного файла, которой с помощью встроенной переменной \$RANDOM генерирует случайную последовательность букв латинского алфавита (рис. 3.14). Дала права доступа на исполнение и затем проверила работу скрипта (рис. 3.15).



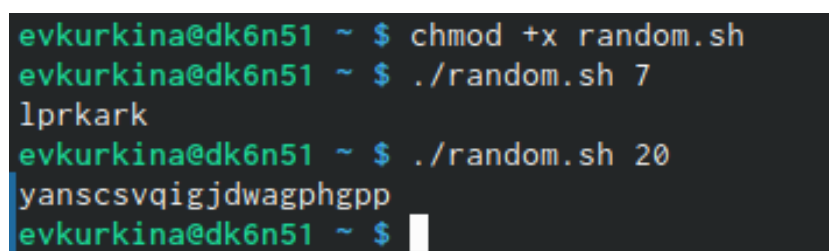
```
evkurkina@dk6n51 ~ $ touch random.sh
evkurkina@dk6n51 ~ $ emacs &
```

Рис. 3.13: Создание нового файла для скрипта 3



```
emacs@dk6n51
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
c=$1
for ((i=0; i<$c; 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. 3.14: Текст скрипта3



```
evkurkina@dk6n51 ~ $ chmod +x random.sh
evkurkina@dk6n51 ~ $ ./random.sh 7
lprkark
evkurkina@dk6n51 ~ $ ./random.sh 20
yanscsvqigjdwagphgpp
evkurkina@dk6n51 ~ $
```

Рис. 3.15: Права доступа результат проверки исполнения файла

4) 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 про

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 1

`seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1

`seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от FIRST до LAST с шагом INCREMENT

`seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности чисел в заданном формате

`seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для STRING для разделения чисел

`seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путей

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

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 дистрибутивов Linux

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,

Скрипты, написанные на `bash`, нельзя запустить на других операционных системах без

4 Выводы

Во время выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы