Лабораторная работа №13

Российский университет дружбы народов

Тимур Андреевич Дарижапов

Содержание

Список таблиц

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

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). Запустим 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. Доработаем программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

Создаём файл lab13-1.sh в редакторе emacs и пишем скрипт(Рисунок 2.1).

```
#!/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. 2.1: Скрипт

Проверяем, предварительно добавив право на выполнение(Рисунок 2.2).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ chmod +x lab13-1.sh
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-1.sh 3 11
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
tadarizhapov@tadarizhapov-VirtualBox:~$
```

Рис. 2.2: Проверка

Изменяем скрипт, чтобы его можно было выполнять в нескольких терминалах(Рисунок 2.3, 2.4).

```
#!/bin/bash
function o {
    s1=$(date +"%s")
    s2=$(date +"%s")
    ((t=$s2-$s1))
    while ((t < t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +"%s")
        ((t=$s2-$s1))
    done
}
function v {
    s1=$(date +"%s")
    s2=$(date +"%s")
    ((t=$s2-$s1))
    while ((t < t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +"%s")
        ((t=$s2-$s1))
     done
```

Рис. 2.3: Изменённый скрипт

```
}
t1=$1
t2=$2
command=$3
while true
do
    if ["$command" == "Выход"]
    then
        echo "Выход"
        exit 0
    fi
    if ["$command" == "Ожидание"]
    then o
    fi
    if ["$command" == "Выполнение"]
    then v
    fi
    echo "Следующее действие: "
    read command
done
```

Рис. 2.4: Изменённый скрипт2

Скрипт работает корректно, но ни одна команда не сработала из-за отказа в доступе(Рисунок 2.5).

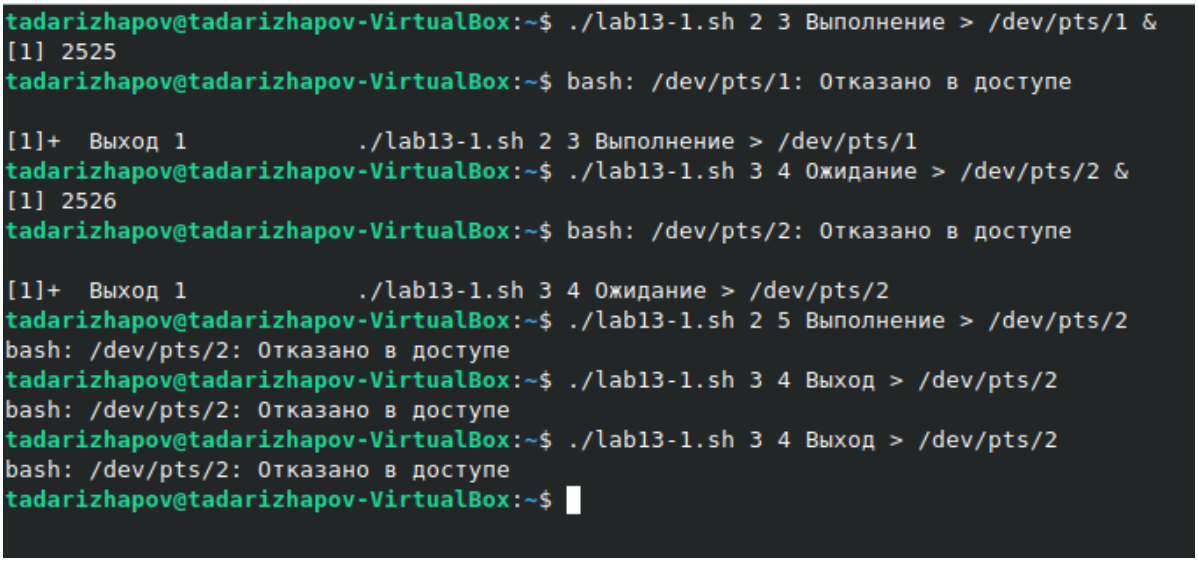

Рис. 2.5: Проверка

2.Реализуем команду man с помощью командного файла. Изучим содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и ко-манд. 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 вкаталоге man1.

Изучим содержимое каталога /usr/share/man/man1(Рисунок 2.6).

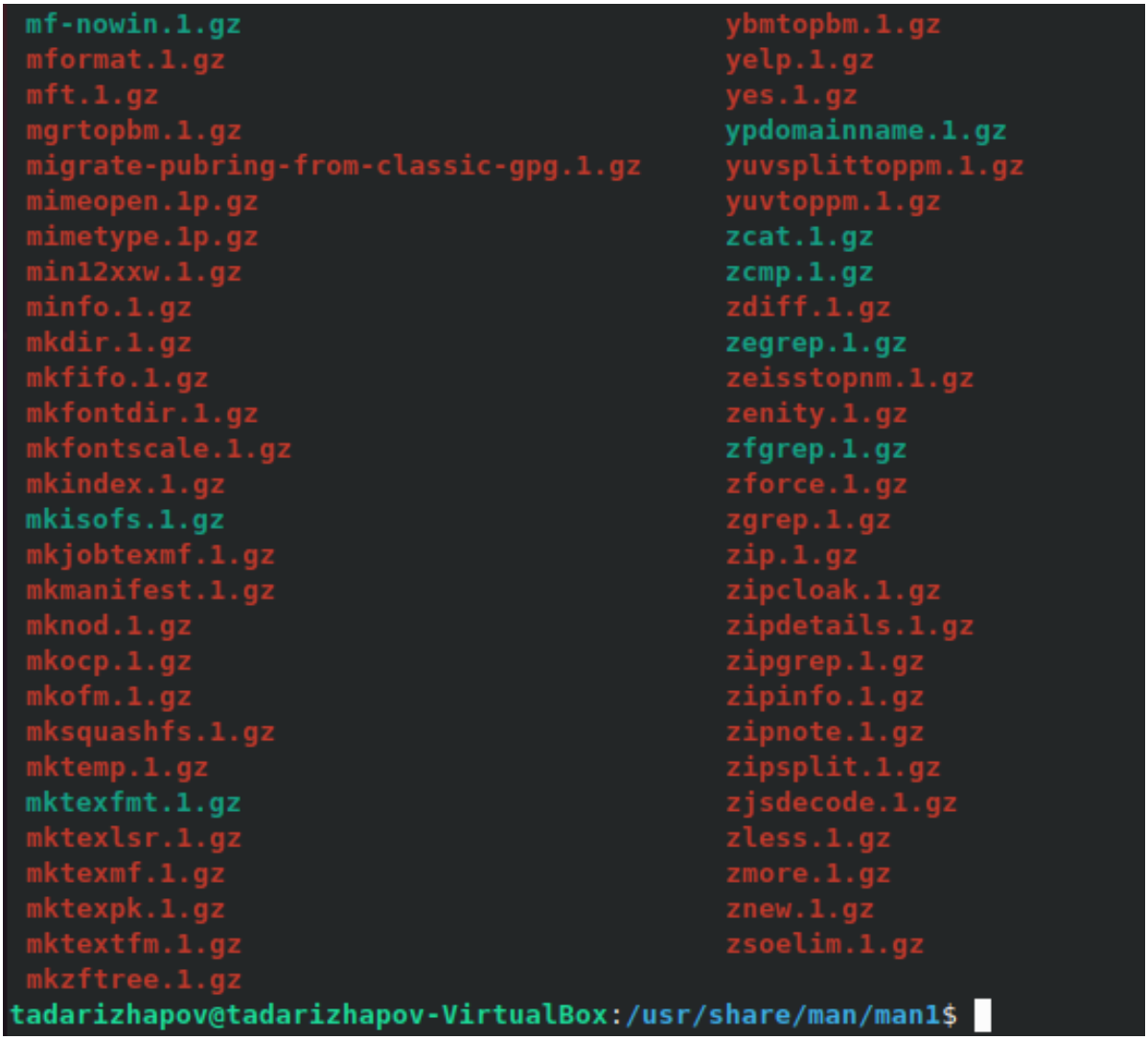

Рис. 2.6: Содержимое каталога

Создаём файл lab13-1.sh в редакторе emacs и пишем скрипт(Рисунок 2.7).

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

Рис. 2.7: Скрипт

Проверка работы скрипта. Перед проверкой добавляем право на выполнение(Рисунок 2.8).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ chmod +x lab13-2.sh
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-2.sh ls
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-2.sh mkdir
tadarizhapov@tadarizhapov-VirtualBox:~$
```

Рис. 2.8: Проверка

Справка команды ls(Рисунок 2.9).

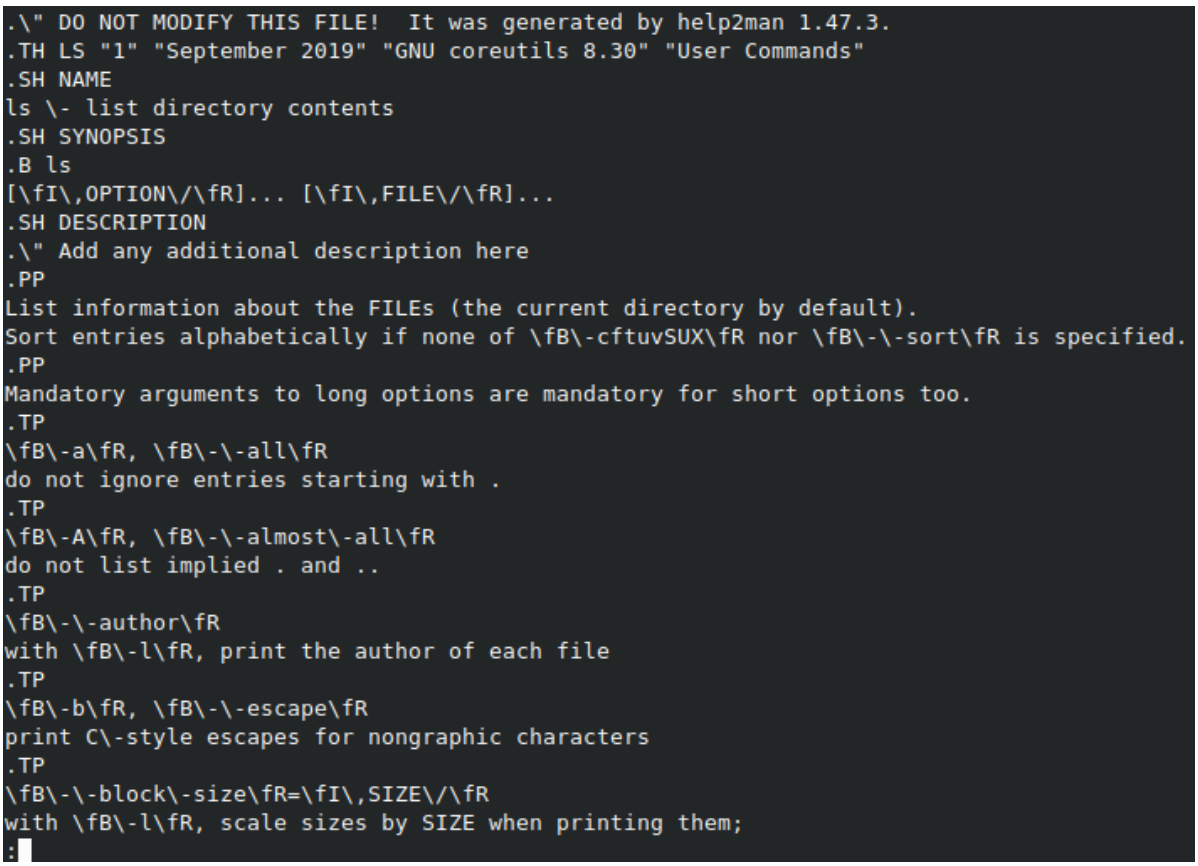

Рис. 2.9: Справка ls

Справка команды mkdir(Рисунок 2.10).

```
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.47.3.
.TH MKDIR "1" "September 2019" "GNU coreutils 8.30" "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
:
```

Рис. 2.10: Справка mkdir

3.Используя встроенную переменную $RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтём, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Создаём файл lab13-3.sh в редакторе emacs и пишем скрипт(Рисунок 2.11).

```
#!/bin/bash
k=$1
for (( i=0; i<$k; 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. 2.11: Скрипт

Проверка работы скрипта. Перед проверкой добавляем право на выполнение(Рисунок 2.12).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ chmod +x lab13-3.sh
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-3.sh 13
doioodxdnutvw
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-3.sh 15
njuaktvhltmsojs
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-3.sh 1
g
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab13-3.sh 1
f
tadarizhapov@tadarizhapov-VirtualBox:~$
```

Рис. 2.12: Проверка

3 Выводы

Я изучил основы программирования в оболочке ОС UNIX. Я научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

4 Ответы на 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 zshot 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 языков общего назначения. Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на быстроте выполнения этого скрипта. Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных .