Отчет по лабораторной работе №13

Дисциплина

Филиппова Анна Дмитриевна

Содержание

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. Создаем файл 1s.sh и пишем соответствующий скрипт. (рис. -fig. 2.1) Пишем 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. -fig. 2.2) (рис. -fig. 2.3)

```
[adfilippova@adfilippova ~]$ touch 1s.sh
[adfilippova@adfilippova ~]$ emacs &
```

Рис. 2.1: Создаем файл

```
#!/bin/bash
t1=$1 # Время ожидания
t2=$2 # Время выполнения
s1=$(date +"%s") # Счетчик времени s1 (сек)
s2=$(date +"%s") # счетчик времени s2 (сек)
((t=$s2-$s1)) # Счетчик времени t (сек). Будет изменяться в цикле
while ((t<t1)) # Цикл ожидания
do
    echo "Ожидание" # Вывод сообщения
    sleep 1 # Пауза на 1 сек для изменения счетчика
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
s1=$(date +"%s") # Обновление счетчика s1
s2=$(date +"%s") # Обновление счетчика s2
((t=$s2-$s1)) # Обновление счетчика t
while ((t<t2)) # Цикл выполнения
do
    echo "Выполнение" # Вывод сообщения
    sleep 1 # Пауза на 1 сек для изменения счетчика
    s2=$(date +"%s")
```

Рис. 2.2: Пишем командный файл

```
s1=$(date +"%s") # Обновление счетчика s1
s2=$(date +"%s") # Обновление счетчика s2
((t=$s2-$s1)) # Обновление счетчика t
while ((t<t2)) # Цикл выполнения
do
    echo "Выполнение" # Вывод сообщения
    sleep 1 # Пауза на 1 сек для изменения счетчика
    s2=$(date +"%s")
    ((t=$s2-$s1))
done
```

Рис. 2.3: Пишем командный файл

Проверяем работу написанного скрипта (команда «./1s.sh 3 5»), предварительно добавив право на исполнение файла (команда «chmod +x 1s.sh»). Скрипт работает корректно.(рис. -fig. 2.4)

```
[adfilippova@adfilippova ~]$ chmod +x 1s.sh
[adfilippova@adfilippova ~]$ ./1s.sh 3 5
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
[adfilippova@adfilippova ~]$
```

Рис. 2.4: Проверка скрипта

После этого изменяем скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверила его работу (команда «./1s.sh 2 5 Ожидание > /dev/pts/2 &» и команда «./1s.sh 2 5 Ожидание > /dev/tty2 »). При этом ни одна из команд не сработала, выводя сообщение “Отказано в доступе”. При этом скрипт работает корректно. (рис. -fig. 2.5) (рис. -fig. 2.6) (рис. -fig. 2.7) (рис. -fig. 2.8)

```
#!/bin/bash
function o
{
    s1=$(date +"%s") # Счетчик времени s1 (сек)
    s2=$(date +"%s") # счетчик времени s2 (сек)
    ((t=$s2-$s1)) # Счетчик времени t (сек). Будет изменяться в цикле
    while ((t<t1)) # Цикл ожидания
    do
        echo "Ожидание" # Вывод сообщения
        sleep 1 # Пауза на 1 сек для изменения счетчика
        s2=$(date +"%s")
        ((t=$s2-$s1))
    done
}
function v
{
    s1=$(date +"%s") # Обновление счетчика s1
    s2=$(date +"%s") # Обновление счетчика s2
    ((t=$s2-$s1)) # Обновление счетчика t
    while ((t<t2)) # Цикл выполнения
    do
```

Рис. 2.5: Изменяем командный файл

```
        echo "Выполнение" # Вывод сообщения
        sleep 1 # Пауза на 1 сек для изменения счетчика
        s2=$(date +"%s")
        ((t=$s2-$s1))
    done
}
t1=$1 # Время ожидания
t2=$2 # Время выполнения
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
```

Рис. 2.6: Изменяем командный файл

```
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. 2.7: Изменяем командный файл

```
[adfilippova@adfilippova ~]$ ./ 1s.sh 2 5 Ожидание > /dev/pts/2 &
[4] 4847
bash: /dev/pts/2: Отказано в доступе
[4]+  Exit 1                  ./ 1s.sh 2 5 Ожидание > /dev/pts/2
[adfilippova@adfilippova ~]$ ./ 1s.sh 2 5 Ожидание > /dev/tty2
bash: /dev/tty2: Отказано в доступе
```

Рис. 2.8: Проверка скрипта

2. Реализуем команду man с помощью командного файла. Изучаем 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. 2.9)

```
[adfilippova@adfilippova ~]$ cd /usr/share/man/man1
[adfilippova@adfilippova man1]$ ls
:.1.gz
[.1.gz
a2p.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-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-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-kernel-data.1.gz
abrt-action-save-package-data.1.gz
```

Рис. 2.9: Изучаем содержимое каталога

Создаем файл 2s.sh и пишем соответствующие скрипт. (рис. -fig. 2.10) (рис. -fig. 2.11)

```
[adfilippova@adfilippova ~]$ touch 2s.sh
[adfilippova@adfilippova ~]$ emacs &
```

Рис. 2.10: Создаем файл

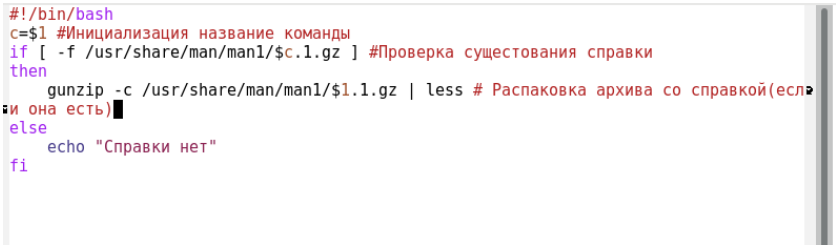

Рис. 2.11: Пишем командный файл

Проверяем работу написанного скрипта (команды «./2s.sh mkdir» и «./2s.sh rm»), предварительно добавив право на исполнение файла (команда «chmod +x 2s.sh»). Скрипт работает корректно. (рис. -fig. 2.12) (рис. -fig. 2.13) (рис. -fig. 2.14)

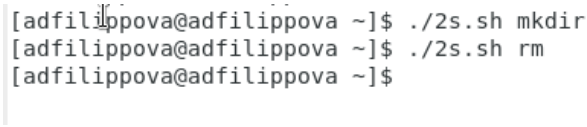

Рис. 2.12: Проверка скрипта

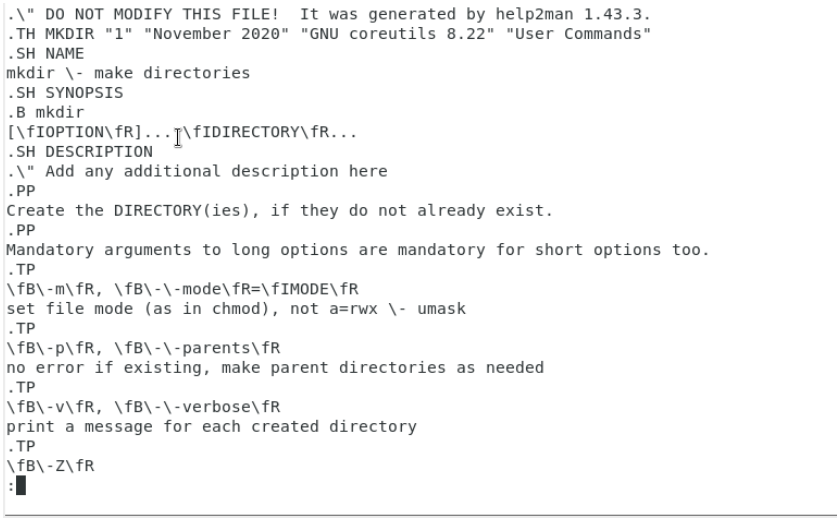

Рис. 2.13: Проверка скрипта

```
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.43.3.
.TH RM "1" "November 2020" "GNU coreutils 8.22" "User Commands"
.SH NAME
rm \- remove files or directories
.SH SYNOPSIS
.B rm
[\fIOPTION\fR]... \fIFILE\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
:
```

Рис. 2.14: Проверка скрипта

3. Создаем файл 3s.sh и пишем соответствующие скрипты. (рис. -fig. 2.15) Используя встроенную переменную $RANDOM, пишем командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. -fig. 2.16)

```
[adfilippova@adfilippova ~]$ touch 3s.sh
[adfilippova@adfilippova ~]$ emacs &
```

Рис. 2.15: Создаем файл

```
#!/bin/bash
c=$1 # Инициализация количество символов
for (( i=0; i<$c; i++)) # Цикл вывода нужного количества символов
do
    (( char=$RANDOM%26+1 )) # Случайные номер от 1 до 26
    case $char in # Вывод символа с помощью оператора выбора
        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. 2.16: Пишем командный файл

Проверяем работу написанного скрипта (команды «./3s.sh 45», «./3s.sh 1000»,

«./3s.sh 1»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh»). Скрипт работает корректно. (рис. -fig. 2.17)

Рис. 2.17: Проверка скрипта

3 Выводы

Я изучила основы программирования в оболочке ОС UNIX, а также научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

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 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 zsh от bash:

- В zsh более быстрое автодополнение для cd с 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 основенеполных 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 действий

5 Библиография

1. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: http://lib.rudn.ru/MegaPro2/Download/MObject/6118.

2. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)

3. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)