

Операционные системы

Лабораторная работа №13

Безрук Мария Андреевна

Содержание

1	Цель работы	3
2	Задание	4
3	Выполнение лабораторной работы	5
4	Контрольные вопросы	10
5	Выводы	13

1 Цель работы

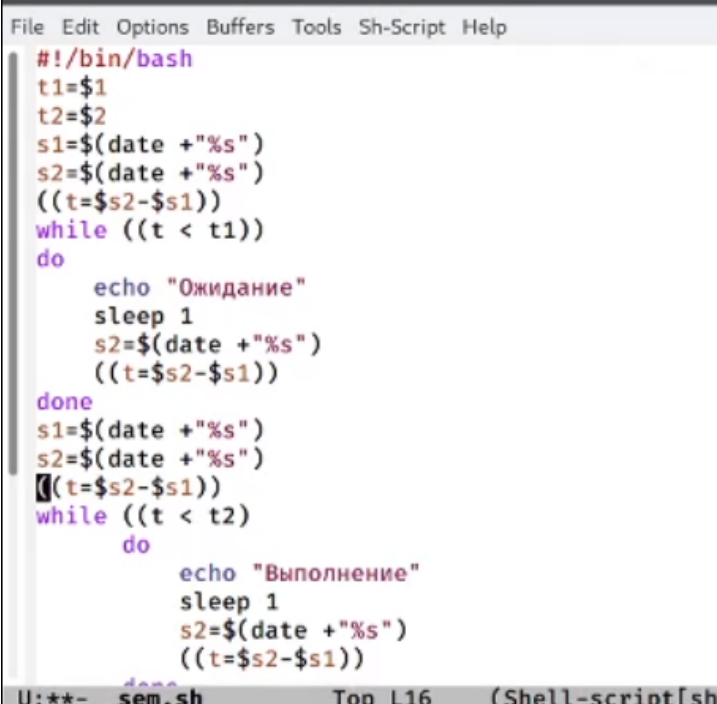
Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров.
2. Реализовать команду `mapс` помощью командного файла.
3. Используя встроенную переменную `“$RANDOM”`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита.

3 Выполнение лабораторной работы

1. Написала 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). Для данной задачи я создала файл: sem.sh и написала соответствующий скрипт.



```
File Edit Options Buffers Tools Sh-Script Help
#!/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:*** sem.sh Top L16 (Shell-script[sh
```

Figure 3.1: Скрипт

Далее я проверила работу написанного скрипта (./sem.sh 4 7), добавив право на исполнение файла (chmod +x .sh). Скрипт работает корректно. Скрипт работает корректно.

```
mmbezruk@dk6n64 ~ $ touch sem.sh
mmbezruk@dk6n64 ~ $ emacs &
[1] 6266
mmbezruk@dk6n64 ~ $ chmod +x sem.sh
mmbezruk@dk6n64 ~ $ ./sem.sh 4 7
Ожидание
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
```

Figure 3.2: Проверка скрипта

После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверила его работу (например, команда «./sem.sh 2 3 Ожидание > /dev/pts/1 &»)

```
#!/bin/bash
function ogidanie
{
    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
```

Figure 3.3: Изменение скрипта

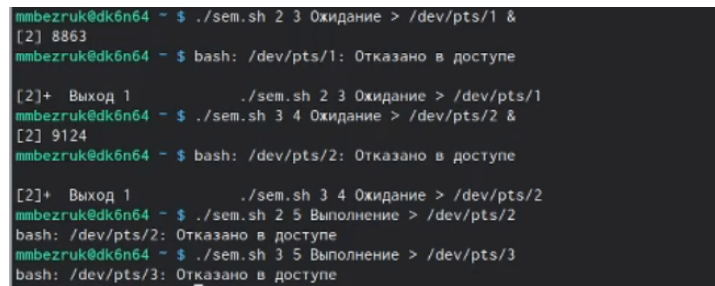
```

while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ogidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие:"
    read command
done

```

Figure 3.4: Изменение скрипта

Но ни одна команда не работала, так как мне “Отказано в доступе”. При этом скрипт работает корректно (команда «./sem.sh »).



```

mmbezruk@dk6n64 ~ $ ./sem.sh 2 3 Ожидание > /dev/pts/1 &
[2] 8863
mmbezruk@dk6n64 ~ $ bash: /dev/pts/1: Отказано в доступе

[2]+  Выход 1          ./sem.sh 2 3 Ожидание > /dev/pts/1
mmbezruk@dk6n64 ~ $ ./sem.sh 3 4 Ожидание > /dev/pts/2 &
[2] 9124
mmbezruk@dk6n64 ~ $ bash: /dev/pts/2: Отказано в доступе

[2]+  Выход 1          ./sem.sh 3 4 Ожидание > /dev/pts/2
mmbezruk@dk6n64 ~ $ ./sem.sh 2 5 Выполнение > /dev/pts/2
bash: /dev/pts/2: Отказано в доступе
mmbezruk@dk6n64 ~ $ ./sem.sh 3 5 Выполнение > /dev/pts/3
bash: /dev/pts/3: Отказано в доступе

```

Figure 3.5: Проверка скрипта

2. Реализовала команду man с помощью командного файла. Изучила 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.

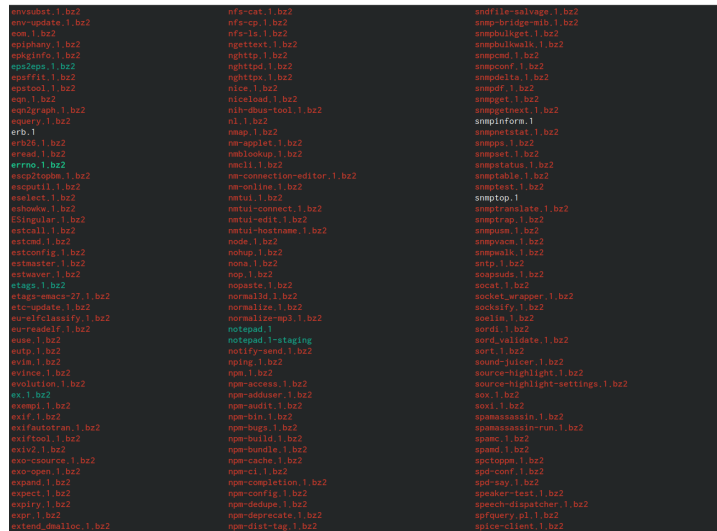


Figure 3.6: Реализация команды

Для данной звдвчи я создала файл : man.sh и написала соответствующий скрипт.

```
#!/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 "Справки по данной команде нет"
fi
```

Figure 3.7: Написание скрипта

Далее я проверила работу написанного скрипта (команды «./man.sh ls», «./man.sh mkdir»), предварительно добавив право на исполнение файла (команда «chmod +x man.sh»). Скрипт сработал и вывел, что по данным командам справок нет. Скрипт работает корректно.

```
mmbezruk@dk6n64 - $ chmod +x man.sh
mmbezruk@dk6n64 - $ ./man.sh ls
Справки по данной команде нет
mmbezruk@dk6n64 - $ ./man.sh mkdir
Справки по данной команде нет
```

Figure 3.8: Проверка скрипта

3. Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создала файл: random.sh и написала соответствующий скрипт.

```
#!/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
```

Figure 3.9: Написание скрипта

Далее я проверила работу написанного скрипта (команды «./random.sh 7» и «./random.sh 15»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh») Скрипт работает корректно.

```
mmbezruk@dk6n64 ~ $ chmod +x random.sh
mmbezruk@dk6n64 ~ $ ./random.sh 7
```

Figure 3.10: Проверка скрипта

```
hjkllmk
```

Figure 3.11: Результат работы скрипта

4 Контрольные вопросы

1) while [\$1 != "exit"] В данной строчке допущены следующие ошибки:

- не хватает пробелов после первой скобки
- 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.

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 cdc помощью Tab
- В zsh существует калькулятор zcalc, способный выполнять вычисления внутри терминала
- 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 языком общего назначения
- Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта
- Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

5 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX, а также научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.