

Отчёт по лабораторной работе №12

дисциплина: Операционные системы

Максим Александрович Мишонков

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	18

Список иллюстраций

4.1	Создание каталога и файла	7
4.2	Скрипт 1	8
4.3	Проверка работы скрипта 1	9
4.4	Изменённый скрипт 1	10
4.5	Изменённый скрипт 1	11
4.6	Проверка работы изменённого скрипта 1	11
4.7	Изучение содержимого файла	12
4.8	Создание файла	12
4.9	Скрипт 2	12
4.10	Проверка работы скрипта 2	13
4.11	Создание файла	13
4.12	Скрипт 3	14
4.13	Проверка работы скрипта 3	15

1 Цель работы

Целью данной лабораторной работы является изучение основ программирования в оболочке ОС UNIX, приобретение навыков написания более сложных командных файлов с использованием логических управляющих конструкций и циклов.

2 Задание

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

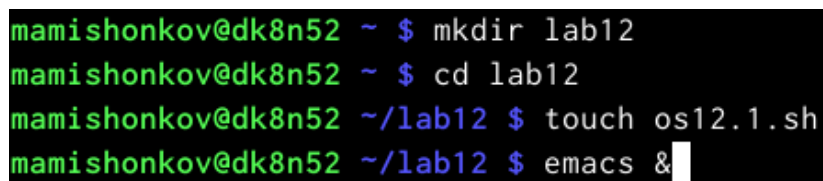
3 Теоретическое введение

Преимущества и недостатки Bash:

Многие языки программирования намного удобнее и понятнее для пользователя. Например, Python более быстр, так как компилируется байтами. Однако главное преимущество Bash - его повсеместное распространение. Более того, Bash позволяет очень легко работать с файловой системой без лишних конструкций. Но относительно такие bash очень сжат. То есть, например, C имеет гораздо более широкие возможности для разработчика.

4 Выполнение лабораторной работы

1. Создал рабочее пространство для скриптов данной лабораторной работы, создал файл для скрипта 1. (рис. [4.1])



```
mamishonkov@dk8n52 ~ $ mkdir lab12
mamishonkov@dk8n52 ~ $ cd lab12
mamishonkov@dk8n52 ~/lab12 $ touch os12.1.sh
mamishonkov@dk8n52 ~/lab12 $ emacs &
```

Рис. 4.1: Создание каталога и файла

2. Написал скрипт 1. (рис. [4.2])

```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидайте"
    sleep 1
    s2=$(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. 4.2: Скрипт 1

3. Проверил работу скрипта 1. (рис. [4.3])

```
mamishonkov@dk8n52 ~/lab12 $ chmod +x os12.1.sh
[1]+  Завершён          emacs
mamishonkov@dk8n52 ~/lab12 $ ./os12.1.sh 4 5
Ожидайте
Ожидайте
Ожидайте
Ожидайте
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
mamishonkov@dk8n52 ~/lab12 $
```

Рис. 4.3: Проверка работы скрипта 1

4. Изменил скрипт 1 так, чтобы его можно было выполнять в нескольких терминалах. (рис. [4.4],[4.5])

```
#!/bin/bash
function ogidania
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-$s1))
    while ((t < t1))
    do
        echo "Ожидайте"
        sleep 1
        s2=$(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
}
```

Рис. 4.4: Изменённый скрипт 1

```

t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" = "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" = "Ожидайте" ]
    then ogidanie
    fi
    if [ "$command" = "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done

```

Рис. 4.5: Изменённый скрипт 1

5. Проверил работу изменённого скрипта 1. (рис. [4.6])

```

mamishonkov@dk8n52 ~/lab12 $ emacs &
[1] 6429
mamishonkov@dk8n52 ~/lab12 $ ./os12.1.sh Ожидайте > /dev/pts/1 &
[2] 9010
[1]   Завершён          emacs
mamishonkov@dk8n52 ~/lab12 $ bash: /dev/pts/1: Отказано в доступе
mamishonkov@dk8n52 ~/lab12 $ ./os12.1.sh Выполнение > /dev/pts/2 &
[3] 9456
[2]   Выход 1           ./os12.1.sh Ожидайте > /dev/pts/1
mamishonkov@dk8n52 ~/lab12 $ bash: /dev/pts/2: Отказано в доступе

```

Рис. 4.6: Проверка работы изменённого скрипта 1

6. Изучил содержимое файла “usr/share/man/man1”. (рис. [4.7])

```

gh-run.1.bz2
gh-run-cancel.1.bz2
gh-run-download.1.bz2
gh-run-list.1.bz2
gh-run-rerun.1.bz2
gh-run-view.1.bz2
gh-run-watch.1.bz2
gh-search.1.bz2
gh-search-issues.1.bz2
gh-search-prs.1.bz2
gh-search-repos.1.bz2
gh-secret.1.bz2
gh-secret-delete.1.bz2
gh-secret-list.1.bz2
gh-secret-set.1.bz2
gh-ssh-key.1.bz2
gh-ssh-key-add.1.bz2
gh-ssh-key-delete.1.bz2
gh-ssh-key-list.1.bz2
gh-status.1.bz2
networkctl.1.bz2
newedge.1.bz2
newgidmap.1.bz2
newgrp.1.bz2
newuidmap.1.bz2
nfs-cat.1.bz2
nfs-cp.1.bz2
nfs-ls.1.bz2
ngettext.1.bz2
nghttp.1.bz2
nghttpd.1.bz2
nghttpx.1.bz2
nice.1.bz2
niceload.1.bz2
nl.1.bz2
nmap.1.bz2
nm-applet.1.bz2
nmblookup.1.bz2
nmcli.1.bz2
nm-connection-editor.1.bz2
mamishonkov@dk8n52 /usr/share/man/man1 $

```

Рис. 4.7: Изучение содержимого файла

7. Создал файл для скрипта 2. (рис. [4.8])

```

mamishonkov@dk8n52 /usr/share/man/man1 $ cd
mamishonkov@dk8n52 ~ $ cd lab12
mamishonkov@dk8n52 ~/lab12 $ touch os12.2.sh
mamishonkov@dk8n52 ~/lab12 $ emacs &

```

Рис. 4.8: Создание файла

8. Написал скрипт 2. (рис. [4.9])

```

#!/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

```

Рис. 4.9: Скрипт 2

9. Проверил работу скрипта 2. (рис. [4.10])

```
mamishonkov@dk8n52 ~/lab12 $ chmod +x os12.2.sh
mamishonkov@dk8n52 ~/lab12 $ ./os12.2.sh ls
Справки по данной команде нет
mamishonkov@dk8n52 ~/lab12 $ ./os12.2.sh mkdir
Справки по данной команде нет
mamishonkov@dk8n52 ~/lab12 $
```

Рис. 4.10: Проверка работы скрипта 2

11. Создал файл для скрипта 3. (рис. [4.11])

```
mamishonkov@dk8n52 ~/lab12 $ touch os12.3.sh
mamishonkov@dk8n52 ~/lab12 $ emacs &
```

Рис. 4.11: Создание файла

10. Написал скрипт 3. (рис. [4.12])

```
#!/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. 4.12: Скрипт 3

11. Проверил работу скрипта 3. (рис. [4.13])

```
mamishonkov@dk8n52 ~/lab12 $ chmod +x os12.3.sh
mamishonkov@dk8n52 ~/lab12 $ ./os12.3.sh 5
wygr
mamishonkov@dk8n52 ~/lab12 $ ./os12.3.sh 10
lecynysup
mamishonkov@dk8n52 ~/lab12 $
```

Рис. 4.13: Проверка работы скрипта 3

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

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

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

`seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации посл

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

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

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` помощью `Tab`

В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри термина

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

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

5 Выводы

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