

# Лабораторная работа 12

## Отчет по лабораторной работе 12

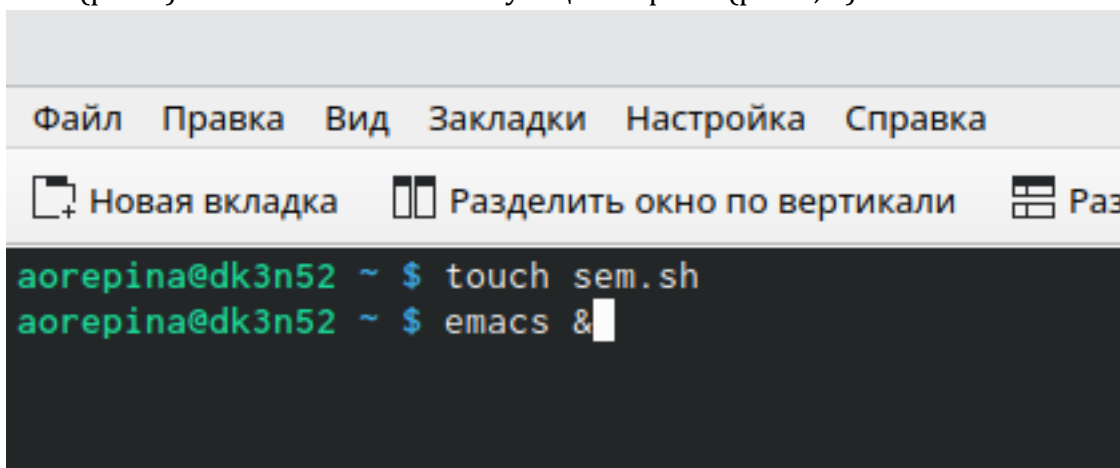
Репина Ангелина Олеговна

### Цель работы

Цель лабораторной работы - изучение основ программирования в оболочке ОС UNIX и получение практических навыков по написанию более сложных командных файлов с использованием логических управляющих конструкций и циклов

### 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` (рис 1) и написала соответствующий скрипт (рис 2, 3)



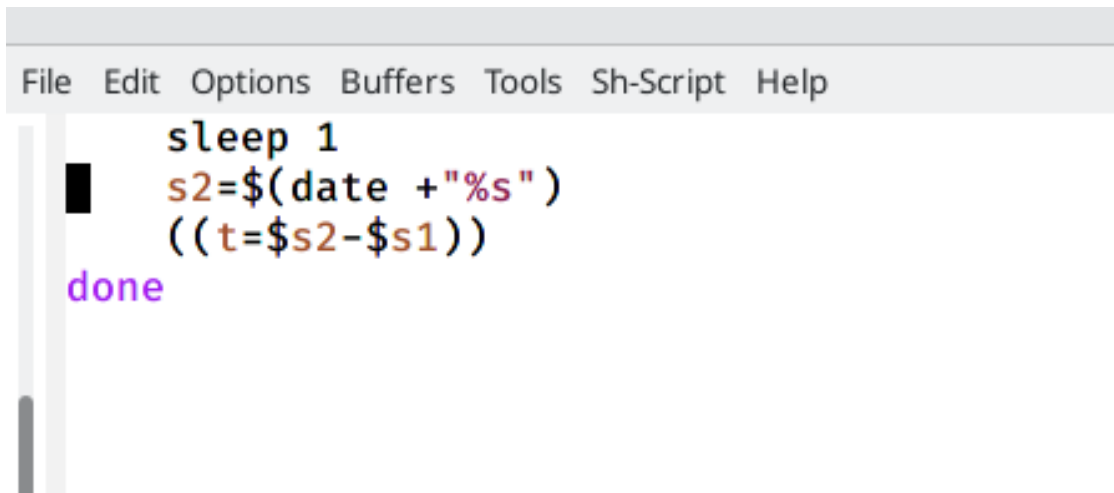
```
Файл  Правка  Вид  Закладки  Настройка  Справка
[+] Новая вкладка  [||] Разделить окно по вертикали  [≡] Раз
aorepina@dk3n52 ~ $ touch sem.sh
aorepina@dk3n52 ~ $ emacs &
```

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))
```

U:--- sem.sh Top L21 (She

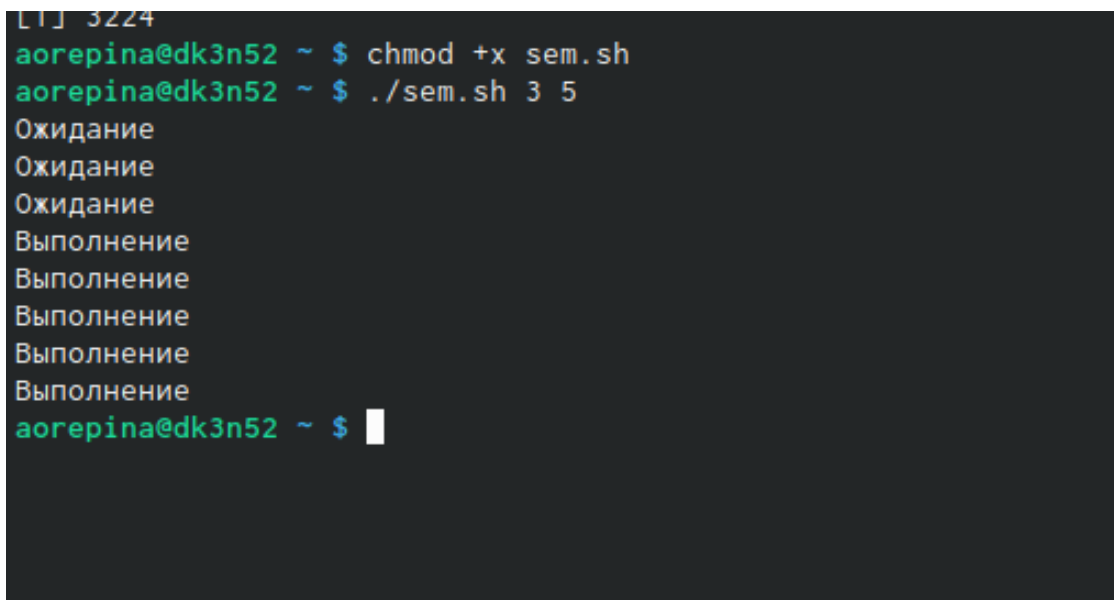
Warning (initialization): An error o

A screenshot of a text editor window with a menu bar containing 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The editor displays a shell script snippet with the following lines: 'sleep 1', 's2=\$(date +%s)', '((t=s2-\$s1))', and 'done'. The text is color-coded: 'sleep' is blue, 's2=' is red, 'date' is green, '%s' is red, 't=' is red, 's2-' is red, 's1' is green, and 'done' is blue. A black cursor is positioned at the end of the first line.

```
File Edit Options Buffers Tools Sh-Script Help
sleep 1
s2=$(date +%s)
((t=s2-$s1))
done
```

3

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

A screenshot of a terminal window with a dark background. The prompt is 'aorepina@dk3n52 ~ \$'. The user enters 'chmod +x sem.sh' and then './sem.sh 3 5'. The script outputs 'Ожидание' (Waiting) three times and 'Выполнение' (Execution) three times. The prompt returns to 'aorepina@dk3n52 ~ \$'.

```
[1] 3224
aorepina@dk3n52 ~ $ chmod +x sem.sh
aorepina@dk3n52 ~ $ ./sem.sh 3 5
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
aorepina@dk3n52 ~ $
```

4

После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах прверила его работу ( например команда ./sem.sh 2 3 Ожидание > /dev/pts/1 &) (рис 5, 6, 7). После проверила работу скрипта и увидела, что мне было отказано в доступе (рис 8)

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
function ozhidanie
{
    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"
    done
}
U:--- sem.sh Top L21 (Shell-script[bash]) Чт ма
Warning (initialization): An error occurred while loading
```

File Edit Options Buffers Tools Sh-Script Help

```
while ((t<t2))  
do  
    echo "Выполнение"  
    sleep 1  
    s2=$(date +%s)  
    ((t=s2-$s1))  
done  
}
```

```
File Edit Options Buffers Tools Sh-Script Help
((t=$s2-$s1))
done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ozhidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done
```

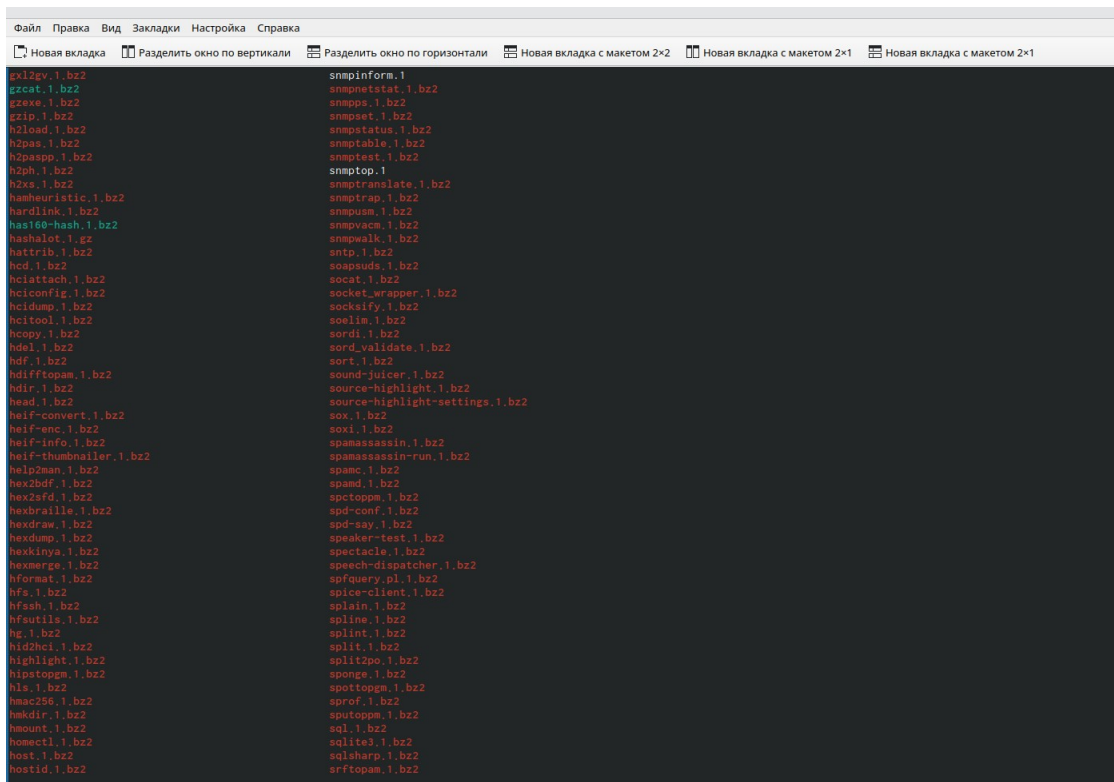
7

```
aorepina@dk3n52 ~ $ ./sem.sh 2 3 Ожидание > /dev/pts/1 &
[1] 4923
aorepina@dk3n52 ~ $ bash: /dev/pts/1: Отказано в доступе

[1]+  Выход 1 ./sem.sh 2 3 Ожидание > /dev/pts/1
aorepina@dk3n52 ~ $ ./sem.sh 2 5 Выполнение > /dev/pts/2 &
[1] 5009
```

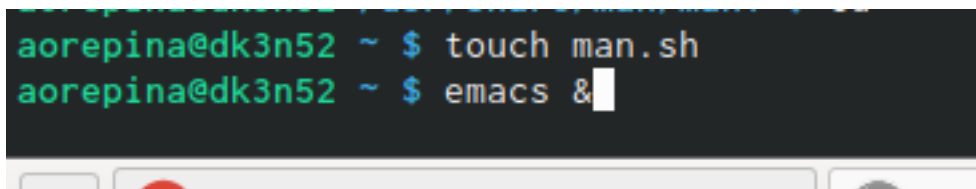
8

- 2) Реализовала команду man с помощью командного файла. Изучила содержимое каталога /usr/share/man/man1 (рис 9). 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



9

Для данной задачи я создала файл man.sh (рис 10) и написала соответствующий скрипт (рис 11)



10

```
File Edit Options Buffers Tools Sh-Script Help
#!/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
```

11

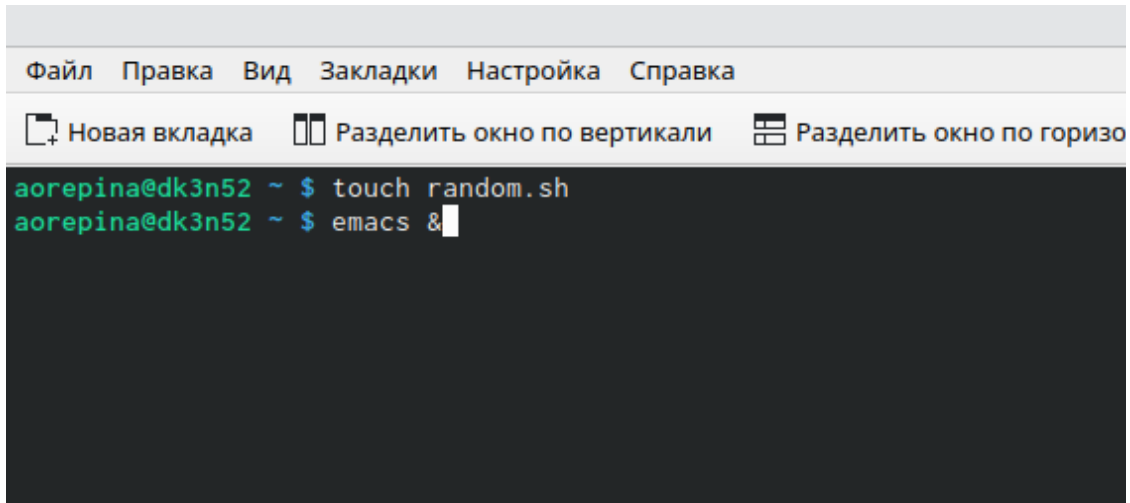
Далее я проверила работу написанного скрипта (./man.sh ls и ./man.sh mkdir) , предварительно добавив право на исполнение файла (chmod +x man.sh) (рис 12). Скрипт работает корректно

```
aorepina@dk3n52 ~ $ chmod +x man.sh
aorepina@dk3n52 ~ $ ./man.sh ls
Справки по данной команде нет
aorepina@dk3n52 ~ $ ./man.sh mkdir
Справки по данной команде нет
aorepina@dk3n52 ~ $
```

12

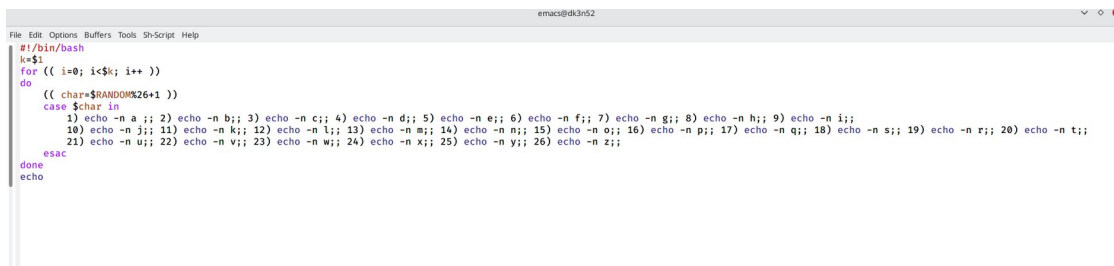
- 3) Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учла, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. Для данной задачи я создала файл random.sh и написала соответствующий скрипт (рис 13, 14)





```
Файл  Правка  Вид  Закладки  Настройка  Справка
+ Новая вкладка  || Разделить окно по вертикали  || Разделить окно по горизон
aorepina@dk3n52 ~ $ touch random.sh
aorepina@dk3n52 ~ $ emacs &
```

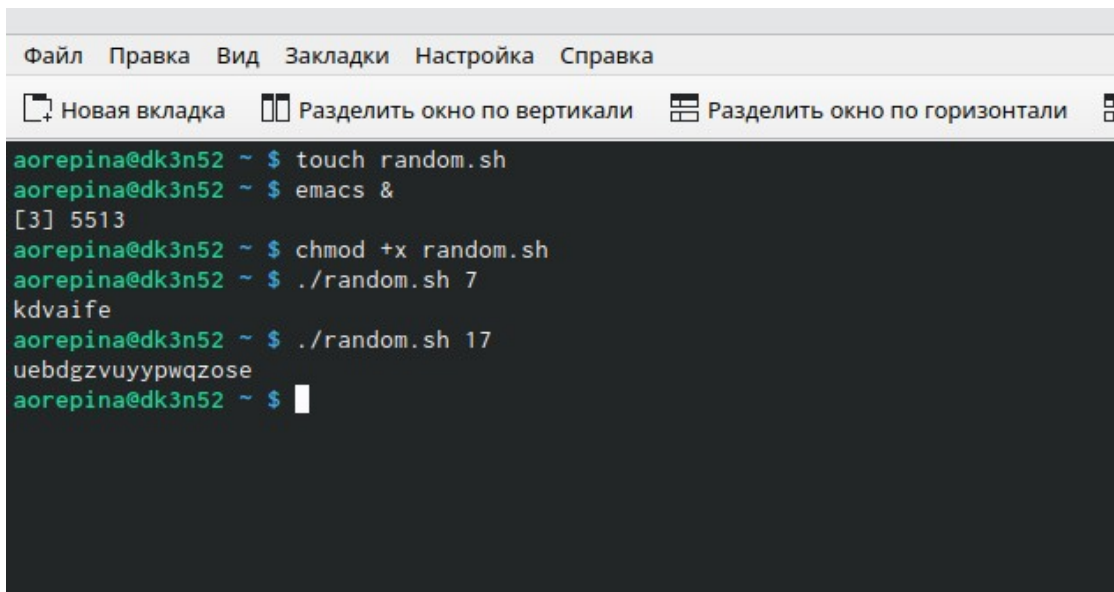
13



```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
k=51
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 ;;
    *) echo -n " " ;;
  esac
done
echo
```

14

Далее я проверила работу написанного скрипта (./random.sh 7; 17), предварительно добавив право на исполнение файла (рис 15). Скрипт работает корректно



```
Файл  Правка  Вид  Закладки  Настройка  Справка
+ Новая вкладка  || Разделить окно по вертикали  || Разделить окно по горизонтали  ||
aorepina@dk3n52 ~ $ touch random.sh
aorepina@dk3n52 ~ $ emacs &
[3] 5513
aorepina@dk3n52 ~ $ chmod +x random.sh
aorepina@dk3n52 ~ $ ./random.sh 7
kdvaife
aorepina@dk3n52 ~ $ ./random.sh 17
uebdgzvuyypwqzose
aorepina@dk3n52 ~ $
```

15

## Выводы

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

## Итоговые 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 ["$1"!= "exit"]` 2). Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: • Первый: `VAR1="Hello," VAR2=" World" VAR3="$VAR2" 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 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 `$((10/3))` будет 3, потому что это целочисленное деление без остатка. 5). Отличия командной оболочки `zsh` от `bash`: • В `zsh` более быстрое автодополнение для `cd` с помощью `Tab` • В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала • В `zsh` поддерживаются числа с плавающей запятой • В `zsh` поддерживаются структуры данных «хэш» • В `zsh` поддерживается раскрытие полного пути на основе неполных данных • В `zsh` поддерживается замена части пути • В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim` 6). `for((a=1; a<= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` 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.