

Презентация к лабораторной работе 11

Дисциплина: Операционные системы

Куликов Максим Игоревич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	12
5	Контрольные вопросы	13

Список таблиц

Список иллюстраций

3.1	Первый скрипт	7
3.2	Работает	8
3.3	Создание второго скрипта	8
3.4	Скрипт №2	8
3.5	Тест второго скрипта	9
3.6	Скрипт №3	9
3.7	Тест №3	10
3.8	Скрипт №4	10
3.9	Тест №4	11

1 Цель работы

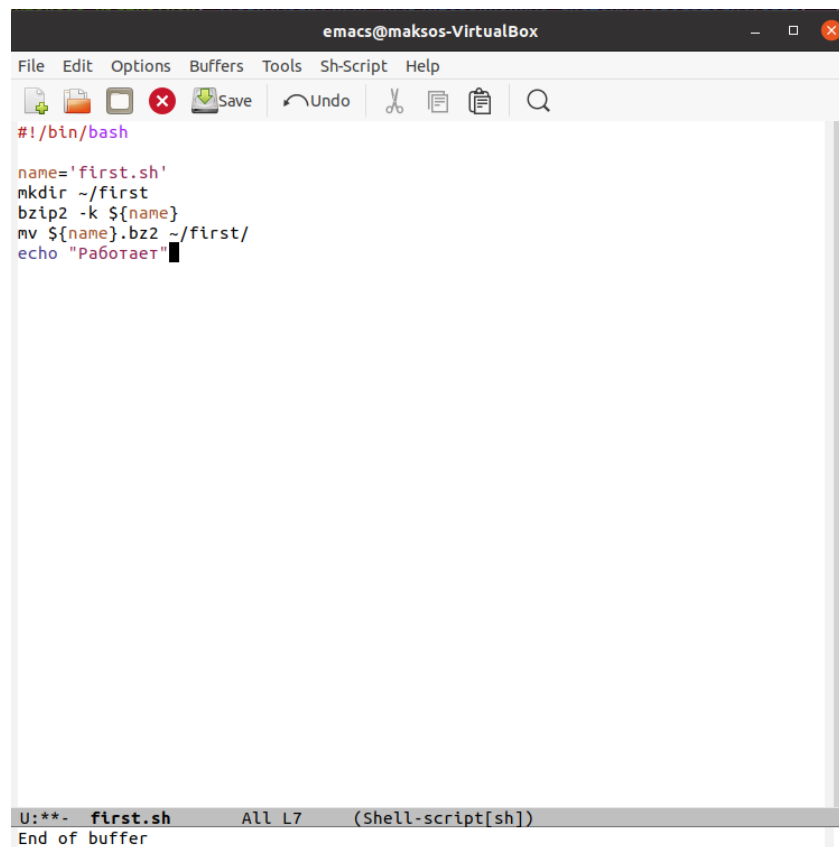
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы

2 Задание

1. Ознакомиться с теоретическим материалом.
2. Выполнить работу.

3 Выполнение лабораторной работы

1. Создаю файл с расширением “sh”. Открываю его через emacs. В нём пишу первый скрипт. (рис. -fig. 3.1)



The screenshot shows the Emacs editor window titled "emacs@maksos-VirtualBox". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". The toolbar contains icons for file operations and editing. The main text area contains the following shell script:

```
#!/bin/bash

name='first.sh'
mkdir ~/first
bzip2 -k ${name}
mv ${name}.bz2 ~/first/
echo "Работает"
```

The status bar at the bottom indicates the current buffer is "U:**- first.sh", the cursor is at "All L7", and the mode is "(Shell-script[sh])".

Рис. 3.1: Первый скрипт

2. Скрипт работает правильно. (рис. -fig. 3.2)



Рис. 3.2: Работает

3. Создаю файл с расширением “sh”. Открываю его через emacs. (рис. -fig. 3.3)

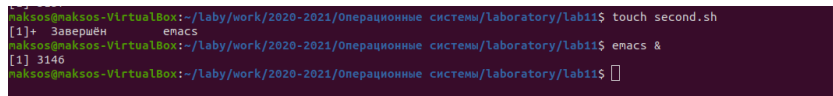


Рис. 3.3: Создание второго скрипта

4. Содержимое второго файла.

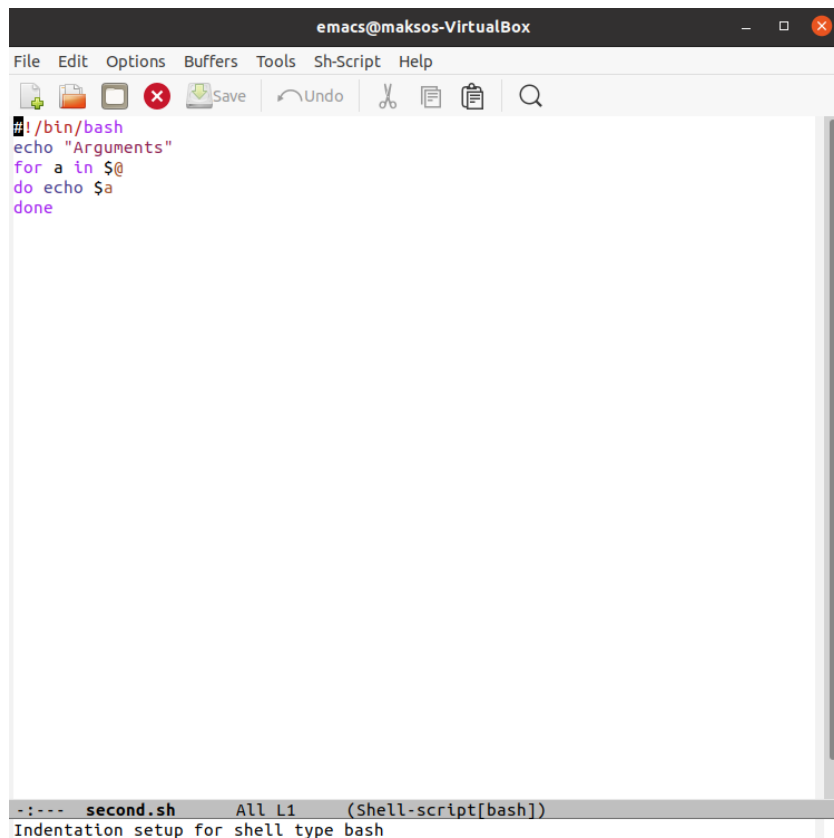


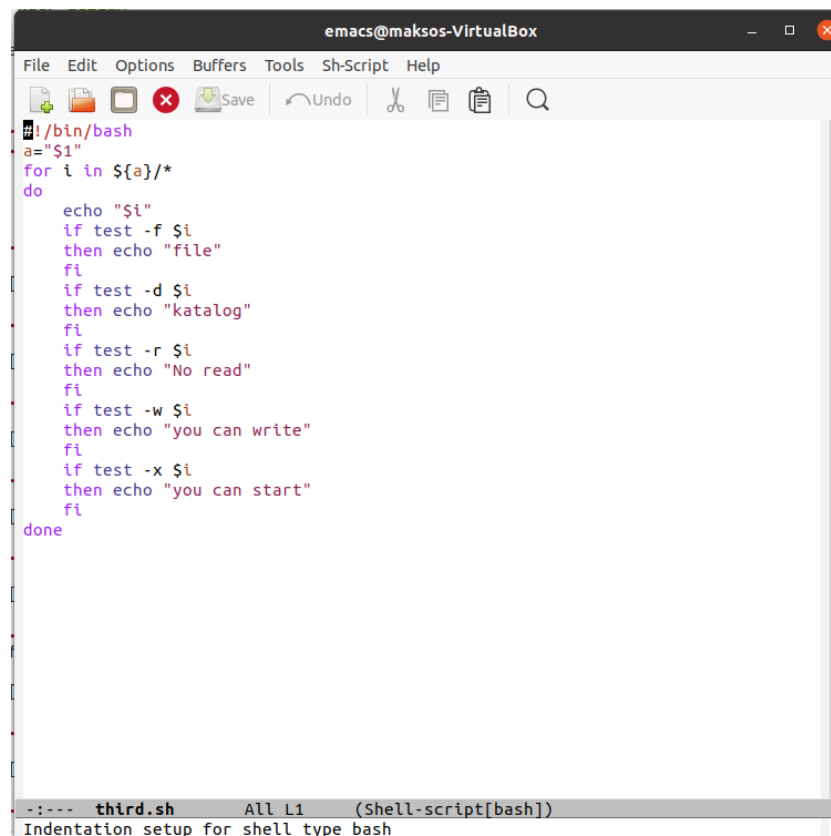
Рис. 3.4: Скрипт №2

5. Разрешаю редактирование скрипта и тестирую его. (рис. -fig. 3.5)

```
maksos@maksos-VirtualBox:~/laby/work/2020-2021/операционные системы/Laboratory/Lab11$ emacs &
[1] 7874
maksos@maksos-VirtualBox:~/laby/work/2020-2021/операционные системы/Laboratory/Lab11$ chmod +x second.sh
[1]+  Завершено      emacs
maksos@maksos-VirtualBox:~/laby/work/2020-2021/операционные системы/Laboratory/Lab11$ ./second.sh 1 2 3 4 5 6 7 8 9 10 11 12
Arguments
1
2
3
4
5
6
7
8
9
10
11
12
maksos@maksos-VirtualBox:~/laby/work/2020-2021/операционные системы/Laboratory/Lab11$
```

Рис. 3.5: Тест второго скрипта

6. Содержимое третьего файла. (рис. -fig. 3.6)



```
emacs@maksos-VirtualBox
File Edit Options Buffers Tools Sh-Script Help
a="/bin/bash"
for i in ${a}/*
do
    echo "$i"
    if test -f $i
    then echo "file"
    fi
    if test -d $i
    then echo "katalog"
    fi
    if test -r $i
    then echo "No read"
    fi
    if test -w $i
    then echo "you can write"
    fi
    if test -x $i
    then echo "you can start"
    fi
done
-:--- third.sh All L1 (Shell-script[bash])
Indentation setup for shell type bash
```

Рис. 3.6: Скрипт №3

7. Тестирую скрипт №3 (рис. -fig. 3.7)

```
maksos@maksos-VirtualBox: ~/laby/work/2020-2021/Операционные системы/laboratory/lab11
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ ls
'-first' first.sh~ image presentation.md second second.sh~ third.sh~
'#first.sh#' first.sh.bz2 Makefile report10.md second.sh third.sh
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ ./third.sh ~
./third.sh: строка 3: синтаксическая ошибка рядом с неожиданным маркером «ln${a}/»
./third.sh: строка 3: `for i in${a}/`
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ emacs &
[1] 21512
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ ./third.sh ~
/home/maksos/image1
katalog
No read
you can write
you can start
/home/maksos/key
file
No read
you can write
/home/maksos/key.pub
file
No read
you can write
/home/maksos/lab10.sh
file
No read
you can write
/home/maksos/lab10.sh~
file
No read
you can write
/home/maksos/laby
katalog
No read
you can write
```

Рис. 3.7: Тест №3

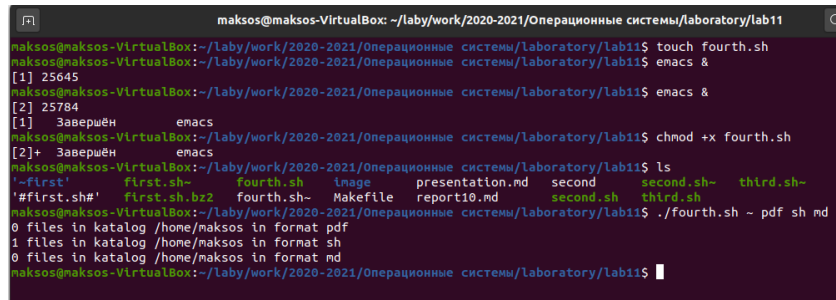
8. Текст 4 программы (рис. -fig. ??)

```
emacs@maksos-VirtualBox
File Edit Options Buffers Tools Sh-Script Help
+ Save Undo Cut Copy Paste Find
#!/bin/bash
b="$1"
shift
for a in $@
do
    l=0
    for i in ${b}/*.$a
    do
        if test -f "$i"
        then
            let l=l+1
        fi
    done
    echo "$l files in katalog $b in format $a"
done

-:--- fourth.sh All L1 (Shell-script[bash])
Indentation setup for shell type bash
```

Рис. 3.8: Скрипт №4

9. Проверяю работоспособность программы (рис. -fig. ??)



```
maksos@maksos-VirtualBox: ~/laby/work/2020-2021/Операционные системы/laboratory/lab11
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ touch fourth.sh
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ emacs &
[1] 25645
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ emacs &
[2] 25784
[1] Завершён      emacs
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ chmod +x fourth.sh
[2]+ Завершён      emacs
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ ls
'-first'  first.sh-  fourth.sh-  image  presentation.md  second  second.sh-  third.sh-
'#first.sh#'  first.sh.bz2  fourth.sh-  Makefile  report10.md  second.sh  third.sh
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$ ./fourth.sh ~ pdf sh md
0 files in katalog /home/maksos in format pdf
1 files in katalog /home/maksos in format sh
0 files in katalog /home/maksos in format md
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab11$
```

Рис. 3.9: Тест №4

4 Выводы

Изучил основы программирования в оболочке ОС UNIX/Linux. Научился писать небольшие командные файлы.

5 Контрольные вопросы

1. Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:
 - ☒ оболочка Борна (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
 - ☒ C-оболочка (или csh) – надстройка на оболочкой Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд;
 - ☒ оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
 - ☒ BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation)
2. POSIX (Portable Operating System Interface for Computer Environments) – набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.
3. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть

выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «`mark=/usr/andy/bin`» присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол `.`, «*`mvafile{mark}`*» переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Оболочка `bash` позволяет работать с массивами. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «`set -A states Delaware Michigan "New Jersey"`» Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.

4. Оболочка `bash` поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляющие собой выражение, подлежащее вычислению. Простейшее выражение – это единичный терм (`term`), обычно целочисленный. Команда `let` берет два операнда и присваивает их переменной. Команда `read` позволяет читать значения переменных со стандартного ввода: «`echo "Please enter Month and Day of Birth ?"`» «`read month day`» В переменные `month` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать её.
5. В языке программирования `bash` можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. В (()) можно записывать условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать резуль-

тат.

7. Стандартные переменные: **PATH**: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога. **PS1** и **PS2**: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >. **HOME**: имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной. **IFS**: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline). **MAIL**: командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение `You have mail` (у Вас есть почта). **TERM**: тип используемого терминала. **LOGNAME**: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.
8. Такие символы, как ' < > * ? | " &, являются метасимволами и имеют для командного процессора специальный смысл.
9. Снятие специального смысла с метасимвола называется экранированием

метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа `\`, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$`, `'`, `,`, `"`. Например, `-echo*` выведет на экран символ `,`, `-echoab'|'cd` выведет на экран строку `ab|*cd`.

10. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `bashкомандный_файл [аргументы]` Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `chmod+химия_файла` Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию.
11. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`.
12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами `test -f [путь до файла]` (для проверки, является ли обычным файлом) и `test -d [путь до файла]` (для проверки, является ли каталогом).
13. Команду `set` можно использовать для вывода списка переменных окружения. В системах `Ubuntu` и `Debian` команда `set` также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения

при работе с данными системами рекомендуется использовать команду «set|more». Команда «typeset» предназначена для наложения ограничений на переменные. Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ \$ является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов \$i, где 0 < i < 10, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i, т.е. аргумента командного файла с порядковым номером i. Использование комбинации символов \$0 приводит к подстановке вместо неё имени данного командного файла.
15. Специальные переменные:
 - ⌘* — ; ? — код завершения последней выполненной команды;
 - ⌘\$\$ — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
 - ⌘! — , ; -- значение флагов командного процессора;
 - ⌘\${#*} — возвращает целое число — количество слов, которые были результатом *; {#name} — возвращает целое значение длины строки в переменной name;
 - ⌘name[n] — n — ; {name[*]} — перечисляет все элементы массива, разделённые пробелом;
 - ⌘name[@] — , ; {name:-value} — если значение переменной name не определено, то оно будет заменено на указанное value;
 - ⌘name : value — ; {name=value} — если name не определено, то ему присваивается значение value;
 - ⌘name?value — , , value; {name+value} — это выражение работает противоположно name — value., value; {name#pattern} — представляет значение переменной name с удалённым самым коротким левым образцом (pattern);
 - ⌘\${#name[*]} и \${#name[@]} — эти выражения возвращают количество элементов в массиве name.