

Отчёта по лабораторной работе №11

Операционный Систем

Коне Сирики НФИБД-01-20

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	19

List of Tables

List of Figures

3.1	рисунок 1	7
3.2	рисунок 2	8
3.3	рисунок 3	8
3.4	рисунок 4	9
3.5	рисунок 5	9
3.6	рисунок 6	10
3.7	рисунок 7	11
3.8	рисунок 8	11
3.9	рисунок 9	12
3.10	рисунок 10	13
3.11	рисунок 11	13
3.12	рисунок 12	14

1 Цель работы

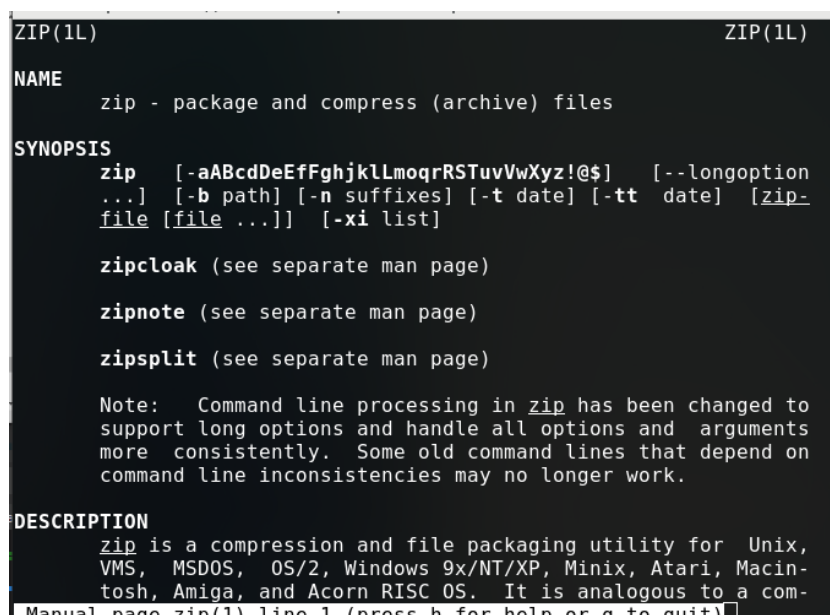
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2 Задание

Командные процессоры (оболочки)

3 Выполнение лабораторной работы

Ход работы: 1. Написал скрипт, который при запуске делает резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моём домашнем каталоге. При этом файл архивируется архиватором zip. Способ использования команд архивации узнала, изучив справку. Команды: `man zip`; `emacs script.sh`; `chmod u+x script.sh`; `./script.sh` (рис. 3.1)

A screenshot of a terminal window displaying the man page for the 'zip' utility. The window has a title bar with 'ZIP(1L)' on both sides. The content is as follows:
NAME
zip - package and compress (archive) files

SYNOPSIS
zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@\$] [--longoption ...] [-b path] [-n suffixes] [-t date] [-tt date] [zip-file [file ...]] [-xi list]

zipcloak (see separate man page)

zipnote (see separate man page)

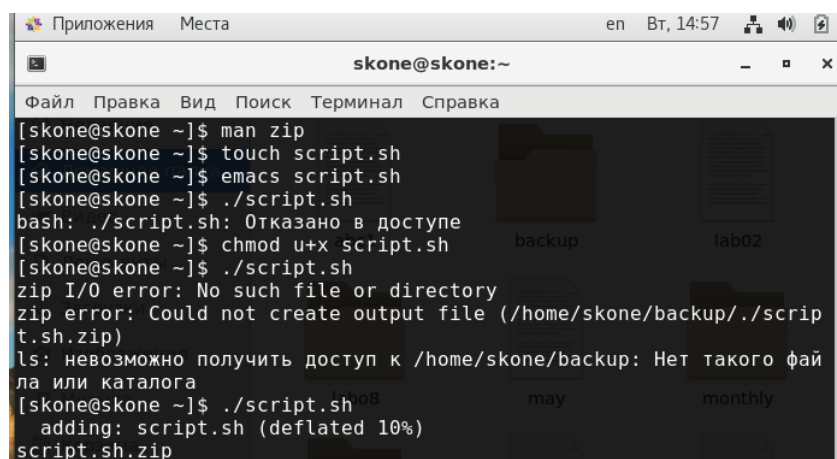
zipsplit (see separate man page)

Note: Command line processing in zip has been changed to support long options and handle all options and arguments more consistently. Some old command lines that depend on command line inconsistencies may no longer work.

DESCRIPTION
zip is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC OS. It is analogous to a com-
Manual page zip(1) line 1 (press h for help or q to quit)

Figure 3.1: рисунок 1

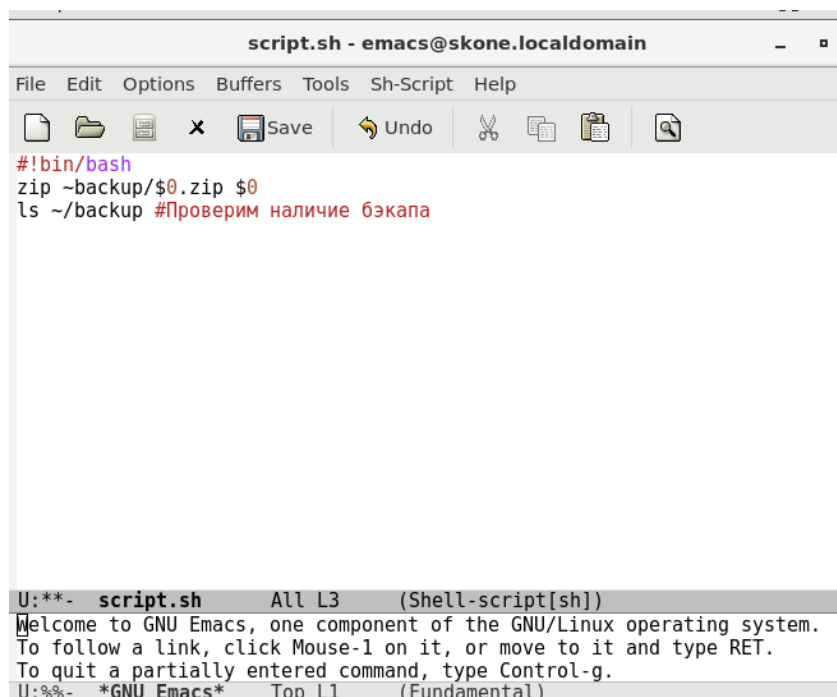
(рис. 3.2)



```
skone@skone:~  
Файл Правка Вид Поиск Терминал Справка  
[skone@skone ~]$ man zip  
[skone@skone ~]$ touch script.sh  
[skone@skone ~]$ emacs script.sh  
[skone@skone ~]$ ./script.sh  
bash: ./script.sh: Отказано в доступе  
[skone@skone ~]$ chmod u+x script.sh  
[skone@skone ~]$ ./script.sh  
zip I/O error: No such file or directory  
zip error: Could not create output file (/home/skone/backup/./script.sh.zip)  
ls: невозможно получить доступ к /home/skone/backup: Нет такого файла или каталога  
[skone@skone ~]$ ./script.sh  
adding: script.sh (deflated 10%)  
script.sh.zip
```

Figure 3.2: рисунок 2

(рис. 3.3)



```
script.sh - emacs@skone.localdomain  
File Edit Options Buffers Tools Sh-Script Help  
[Icons]  
#!/bin/bash  
zip ~backup/$0.zip $0  
ls ~/backup #Проверим наличие бэкапа  
  
U:**- script.sh All L3 (Shell-script[sh])  
Welcome to GNU Emacs, one component of the GNU/Linux operating system.  
To follow a link, click Mouse-1 on it, or move to it and type RET.  
To quit a partially entered command, type Control-g.  
U:%%- *GNU Emacs* Top L1 (Fundamental)
```

Figure 3.3: рисунок 3

(рис. 3.4)

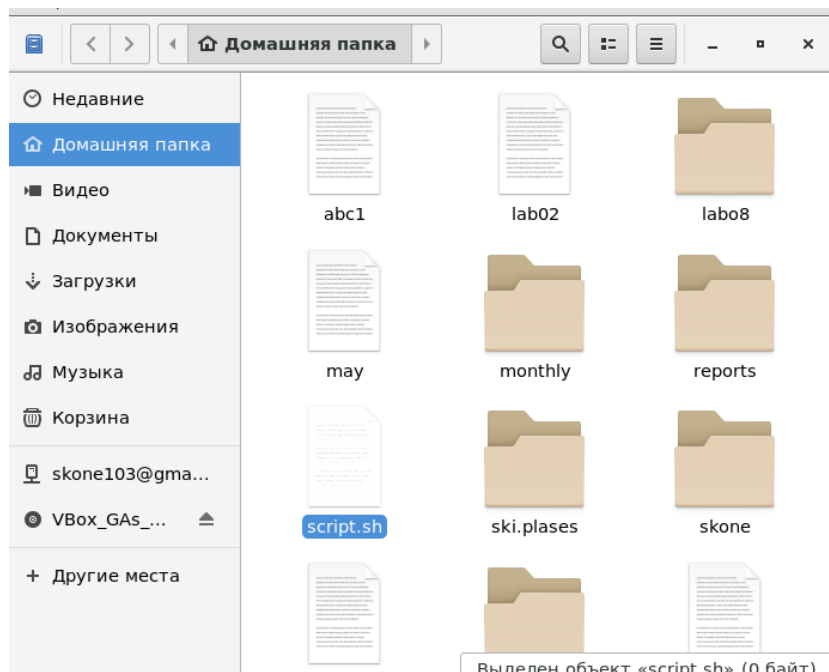


Figure 3.4: рисунок 4

2. Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт может последовательно распечатывать значения всех переданных аргументов. Команды: `emacs script2.sh`; `chmod u+x script2.sh`; `./script2.sh` (рис. 3.5)

(рис. 3.5)

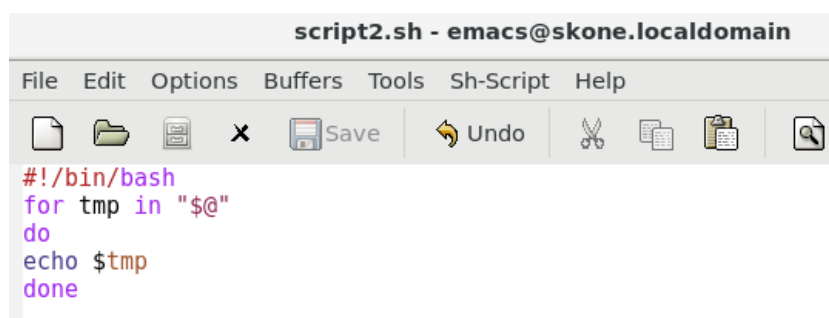


Figure 3.5: рисунок 5

(рис. 3.6)

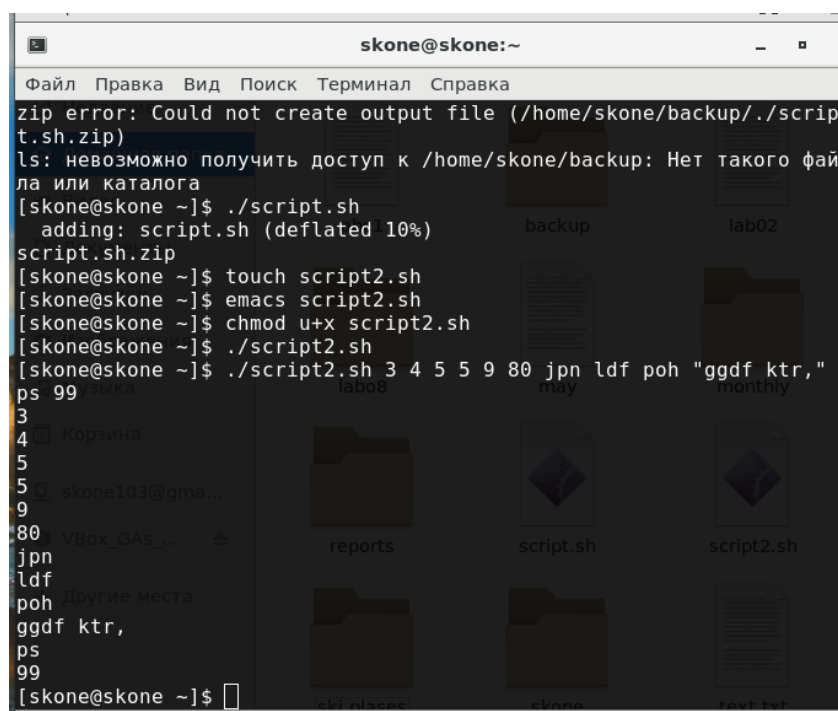
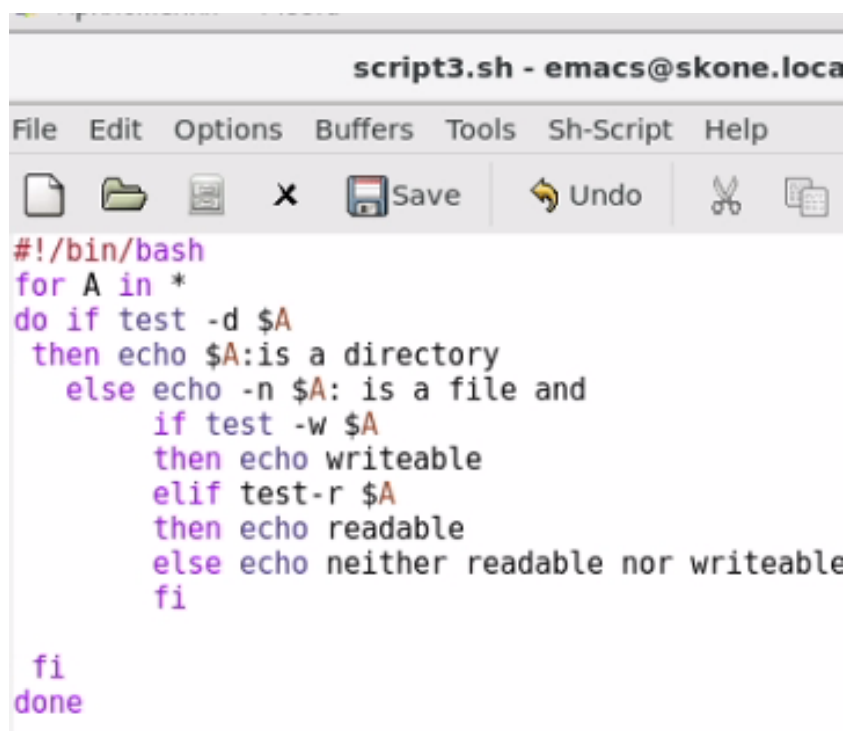


Figure 3.6: рисунок 6

3. Написал командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`).

Он выдаёт информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога.

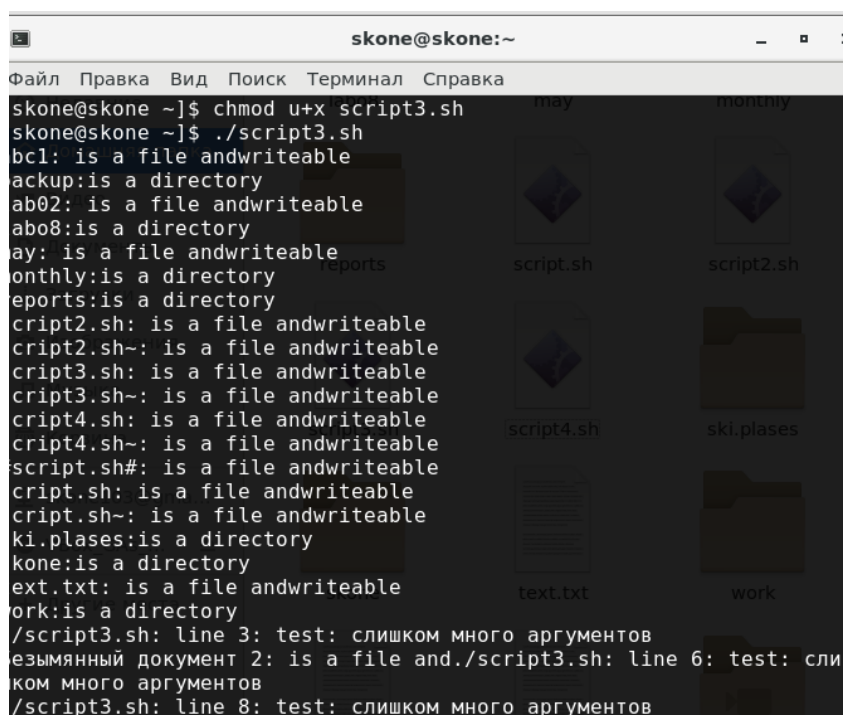
Команды: `emacs script3.sh`; `chmod u+x script3.sh`; `./script3.sh` Загрузки (рис. 3.7)



```
#!/bin/bash
for A in *
do if test -d $A
then echo $A:is a directory
else echo -n $A: is a file and
    if test -w $A
    then echo writeable
    elif test -r $A
    then echo readable
    else echo neither readable nor writeable
    fi
fi
done
```

Figure 3.7: рисунок 7

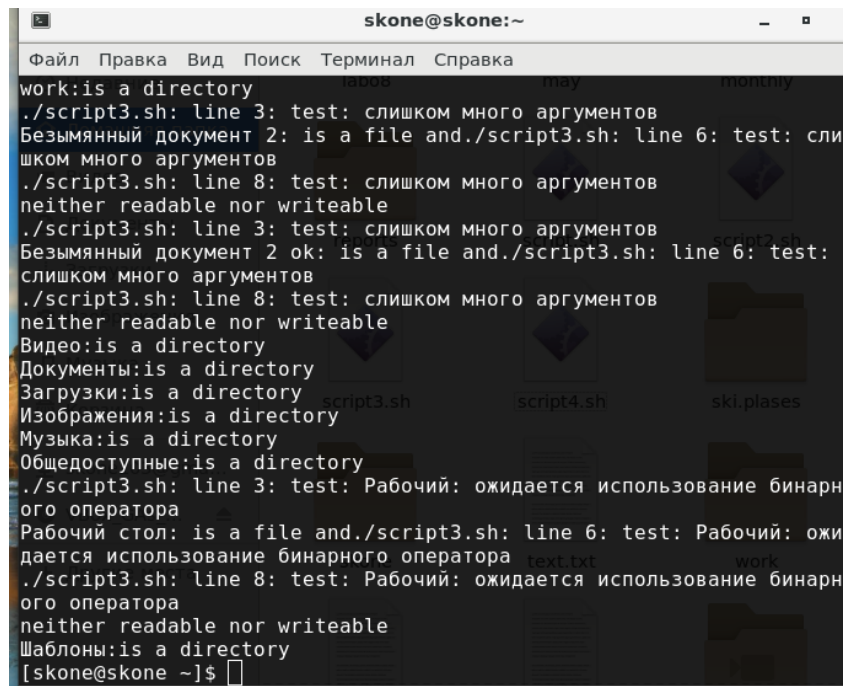
(рис. 3.8)



```
skone@skone ~]$ chmod u+x script3.sh
skone@skone ~]$ ./script3.sh
bcl1: is a file andwriteable
ackup:is a directory
ab02: is a file andwriteable
abo8:is a directory
ay: is a file andwriteable
onthly:is a directory
reports:is a directory
cript2.sh: is a file andwriteable
cript2.sh~: is a file andwriteable
cript3.sh: is a file andwriteable
cript3.sh~: is a file andwriteable
cript4.sh: is a file andwriteable
cript4.sh~: is a file andwriteable
script.sh#: is a file andwriteable
cript.sh: is a file andwriteable
cript.sh~: is a file andwriteable
ki.plases:is a directory
kone:is a directory
ext.txt: is a file andwriteable
ork:is a directory
./script3.sh: line 3: test: слишком много аргументов
языманный документ 2: is a file and./script3.sh: line 6: test: сли
ком много аргументов
./script3.sh: line 8: test: слишком много аргументов
```

Figure 3.8: рисунок 8

(рис. 3.9)



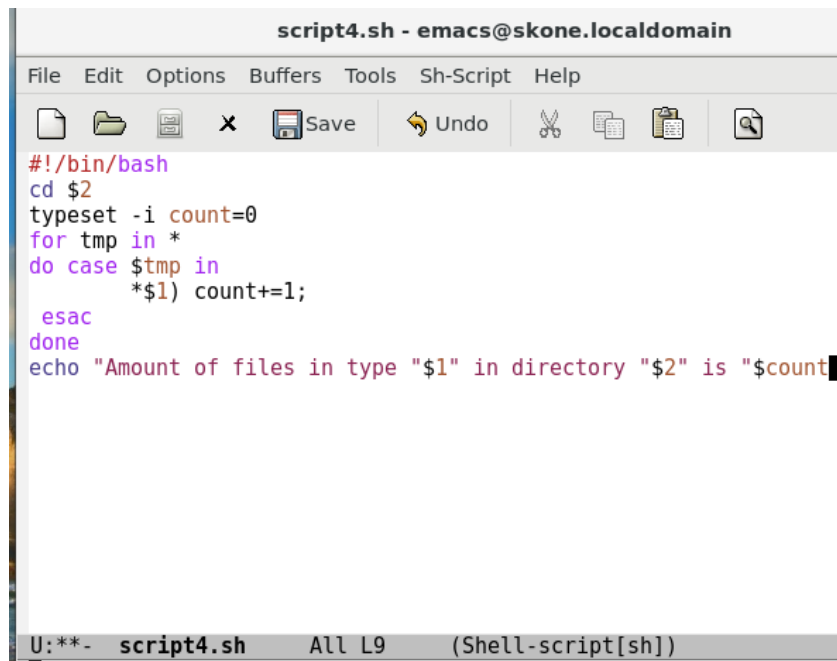
```
skone@skone:~  
Файл Правка Вид Поиск Терминал Справка  
work:is a directory  
./script3.sh: line 3: test: слишком много аргументов  
Безымянный документ 2: is a file and./script3.sh: line 6: test: сли  
шком много аргументов  
./script3.sh: line 8: test: слишком много аргументов  
neither readable nor writeable  
./script3.sh: line 3: test: слишком много аргументов  
Безымянный документ 2 ok: is a file and./script3.sh: line 6: test:  
слишком много аргументов  
./script3.sh: line 8: test: слишком много аргументов  
neither readable nor writeable  
Видео:is a directory  
Документы:is a directory  
Загрузки:is a directory  
Изображения:is a directory  
Музыка:is a directory  
Общедоступные:is a directory  
./script3.sh: line 3: test: Рабочий: ожидается использование бинарн  
ого оператора  
Рабочий стол: is a file and./script3.sh: line 6: test: Рабочий: ожи  
дается использование бинарного оператора  
./script3.sh: line 8: test: Рабочий: ожидается использование бинарн  
ого оператора  
neither readable nor writeable  
Шаблоны:is a directory  
[skone@skone ~]$
```

Figure 3.9: рисунок 9

4. Написал командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки. (рис. 3.10)

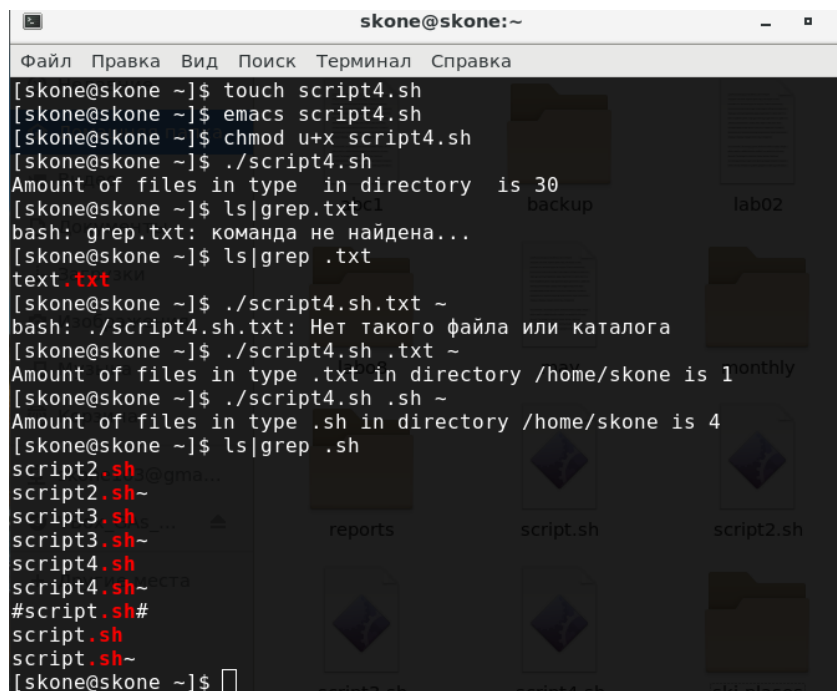
Команды: emacs script4.sh; chmod u+x script4.sh; ./script4.sh .txt ~; ls|grep .txt (проверка); ./script4.sh .sh ~; ls|grep .sh (проверка); ./04.sh .cpp Загрузки; ls Загрузки|grep .cpp (рис. 3.11

(рис. 3.12)



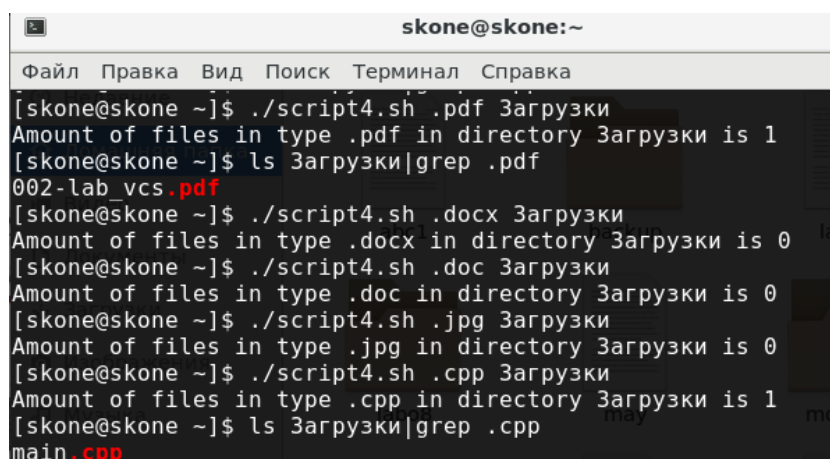
```
#!/bin/bash
cd $2
typeset -i count=0
for tmp in *
do case $tmp in
    *$1) count+=1;
    esac
done
echo "Amount of files in type \"$1\" in directory \"$2\" is \"$count"
```

Figure 3.10: рисунок 10



```
[skone@skone ~]$ touch script4.sh
[skone@skone ~]$ emacs script4.sh
[skone@skone ~]$ chmod u+x script4.sh
[skone@skone ~]$ ./script4.sh
Amount of files in type in directory is 30
[skone@skone ~]$ ls|grep.txt
bash: grep.txt: команда не найдена...
[skone@skone ~]$ ls|grep .txt
text.txt
[skone@skone ~]$ ./script4.sh.txt ~
bash: ./script4.sh.txt: Нет такого файла или каталога
[skone@skone ~]$ ./script4.sh .txt ~
Amount of files in type .txt in directory /home/skone is 1
[skone@skone ~]$ ./script4.sh .sh ~
Amount of files in type .sh in directory /home/skone is 4
[skone@skone ~]$ ls|grep .sh
script2.sh
script2.sh~
script3.sh
script3.sh~
script4.sh
script4.sh~
#script.sh#
script.sh
script.sh~
[skone@skone ~]$
```

Figure 3.11: рисунок 11



```
skone@skone:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
[skone@skone ~]$ ./script4.sh .pdf Загрузки  
Amount of files in type .pdf in directory Загрузки is 1  
[skone@skone ~]$ ls Загрузки|grep .pdf  
002-lab_vcs.pdf  
[skone@skone ~]$ ./script4.sh .docx Загрузки  
Amount of files in type .docx in directory Загрузки is 0  
[skone@skone ~]$ ./script4.sh .doc Загрузки  
Amount of files in type .doc in directory Загрузки is 0  
[skone@skone ~]$ ./script4.sh .jpg Загрузки  
Amount of files in type .jpg in directory Загрузки is 0  
[skone@skone ~]$ ./script4.sh .cpp Загрузки  
Amount of files in type .cpp in directory Загрузки is 1  
[skone@skone ~]$ ls Загрузки|grep .cpp  
main.cpp
```

Figure 3.12: рисунок 12

Контрольные вопросы:

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются? Командная оболочка — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: 1)Оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; 2)С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С- подобный синтаксис команд с возможностью сохранения истории выполнения команд; 3)Оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна; 4)BASH — сокращение от Bourne Again Shell, в основе своей совмещает свойства оболочек С и Корна
2. Что такое POSIX? POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны 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. Как определяются переменные и массивы в языке программирования `bash`?
Переменные, используемые в `bash`, имеют тип строка символов. Их не нужно объявлять специально, можно просто задать в любом месте программы. Массив – это также массив строк, который создаётся командой `set` – А список значений через пробел. Использовать и задавать элементы массива можно, используя индексы: массив[65]=что-то
4. Каково назначение операторов `let` и `read`? Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Команда `read` позволяет читать значения переменных со стандартного ввода.
5. Какие арифметические операции можно применять в языке программирования `bash`? Можно применять все простые арифметические операции, а также операции булевой алгебры
6. Что означает операция `(())`? В двойных скобках записываются условия
7. Какие стандартные имена переменных Вам известны? Значением переменной `PATH` (т.е. `$PATH`) является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа `/`. Переменные `PS1` и `PS2` предназначены для отображения промптера командного процессора. `PS1` — это промптер командного процессора, по умолчанию его значение равно символу `$` 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 (`new line`). `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`. `TERM` — тип используемого терминала. `LOGNAME` — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему

8. Что такое метасимволы? Такие символы, как `' < > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл. Например, `*` в имени файла означает любую последовательность символов.
9. Как экранировать метасимволы? Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа `.` Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$, ', , "`.
10. Как создавать и запускать командные файлы? Командный файл — это текстовый файл, в котором записана последовательность команд. Для `bash` они имеют расширение `.sh`. Для запуска командного файла нужно ввести в командной строке: `bash командный_файл [аргументы]` Или, чтобы не вводить каждый раз `bash`, сделать этот файл исполняемым и вводить просто: `командный_файл [аргументы]`
11. Как определяются функции в языке программирования `bash`? Функция — это некоторая группа команд. Для её создания существует ключевое слово

function, после которого следует имя функции и список команд, заключённых в фигурные скобки.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом? Это можно сделать командой `test -d file`. Если `file` – каталог, команда вернёт значение истина. Если файл – ложь.
13. Каково назначение команд `set`, `typeset` и `unset`? Команда `set` изменяет значения внутренних переменных сценария. Команда `unset` удаляет переменную, фактически устанавливает ее значение `null`. Команда `typeset` задает и/или накладывает ограничения на переменные.
14. Как передаются параметры в командные файлы? 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.
15. Назовите специальные переменные языка `bash` и их назначение

`$1-9` или `${10-...}` – полученные параметры. `$#` – число параметров, указанных в командной строке при вызове данного командного файла на выполнение. `$*` – отображается вся командная строка или параметры оболочки; `$?` – код завершения последней выполненной команды; `$$` – уникальный идентификатор процесса, в рамках которого выполняется командный процессор; `$!` – номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; `$-` – значение флагов командного процессора; `${#}` – *возвращает целое число — количество слов, которые были результатом* `$`; `${#name}` – возвращает целое значение длины строки в переменной `name`; `${name[n]}` – обращение к `n`-му элементу массива; `${name[*]}` – перечисляет все элементы массива, разделённые пробелом; `${name[@]}` – то же самое, но

позволяет учитывать символы пробелы в самих переменных; `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`; `${name:value}` — проверяется факт существования переменной; `${name=value}` — если `name` не определено, то ему присваивается значение `value`; `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке; `${name+value}` — это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `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`.

4 Выводы

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