

Лабораторная работа-10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Доленко Дарья Васильевна НБИбд-01-21

Содержание

1 Цель работы	4
2 Выполнение лабораторной работы	5
3 Вывод	13
4 Ответы на контрольные вопросы:	14

Список иллюстраций

2.1	создание файла и изменение его прав	5
2.2	справка по архиватору	6
2.3	скрипт	6
2.4	запускаю	7
2.5	создан архив	7
2.6	скрипт	8
2.7	запуск и проверка	8
2.8	скрипт	9
2.9	запуск команды	10
2.10	скрипт	11
2.11	запуск и проверка	12

1 Цель работы

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

2 Выполнение лабораторной работы

Пишу скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в домашнем каталоге. При этом файл архивируется архиватором zip. Способ использования команд архивации необходимо было узнать, изучив справку. (рис. [fig. 2.1]2.2[fig. 2.3]2.42.5)

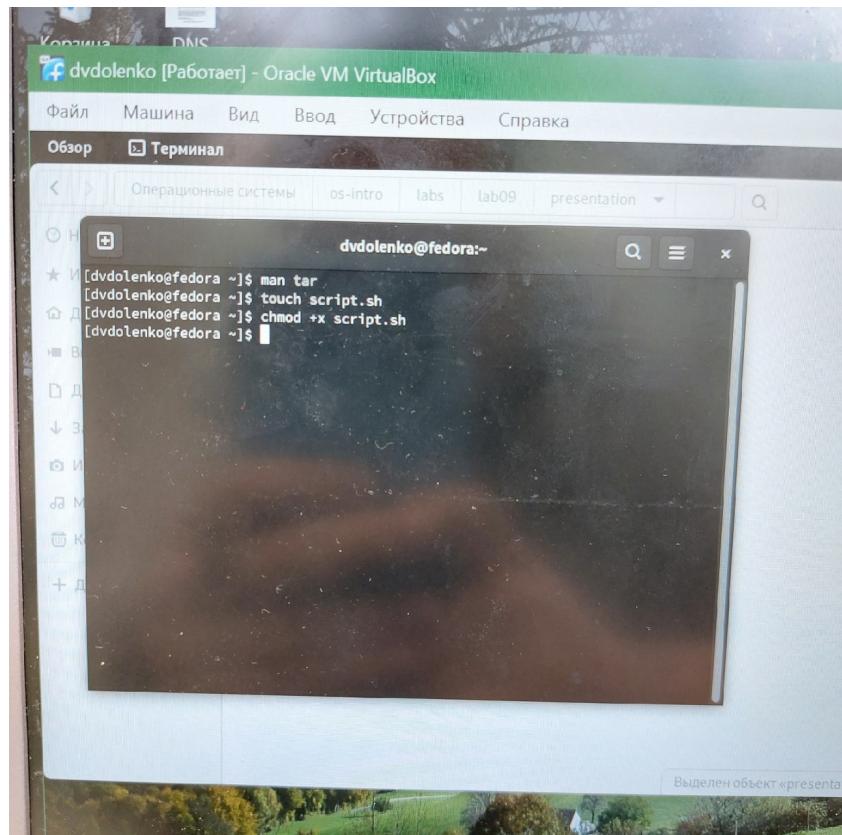
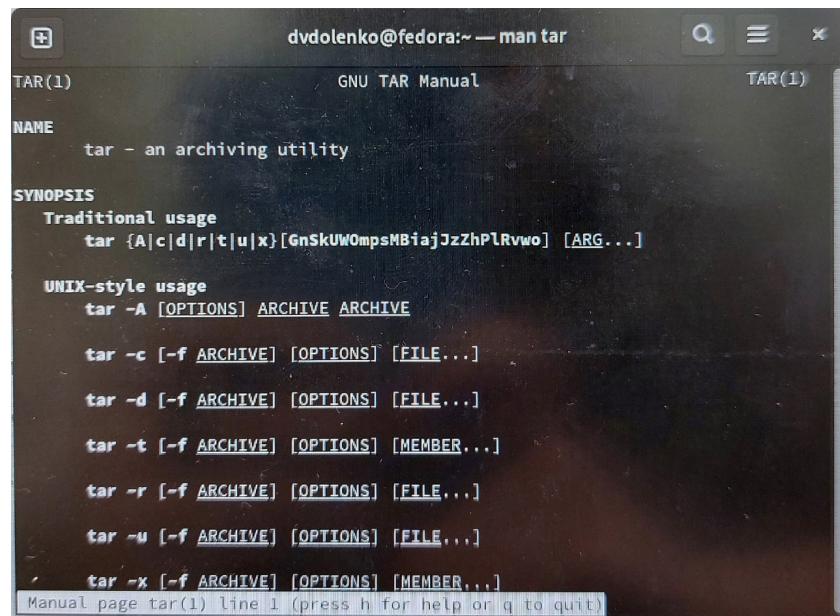


Рис. 2.1: создание файла и изменение его прав



dvdolenko@fedora:~ — man tar

TAR(1) GNU TAR Manual TAR(1)

NAME

tar — an archiving utility

SYNOPSIS

Traditional usage

```
tar {A|c|d|r|t|u|x} [GnSkUW0mpsMBiajJzZhPlRvw] [ARG...]
```

UNIX-style usage

```
tar -A [OPTIONS] ARCHIVE ARCHIVE
```

```
tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
```

```
tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
```

```
tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
```

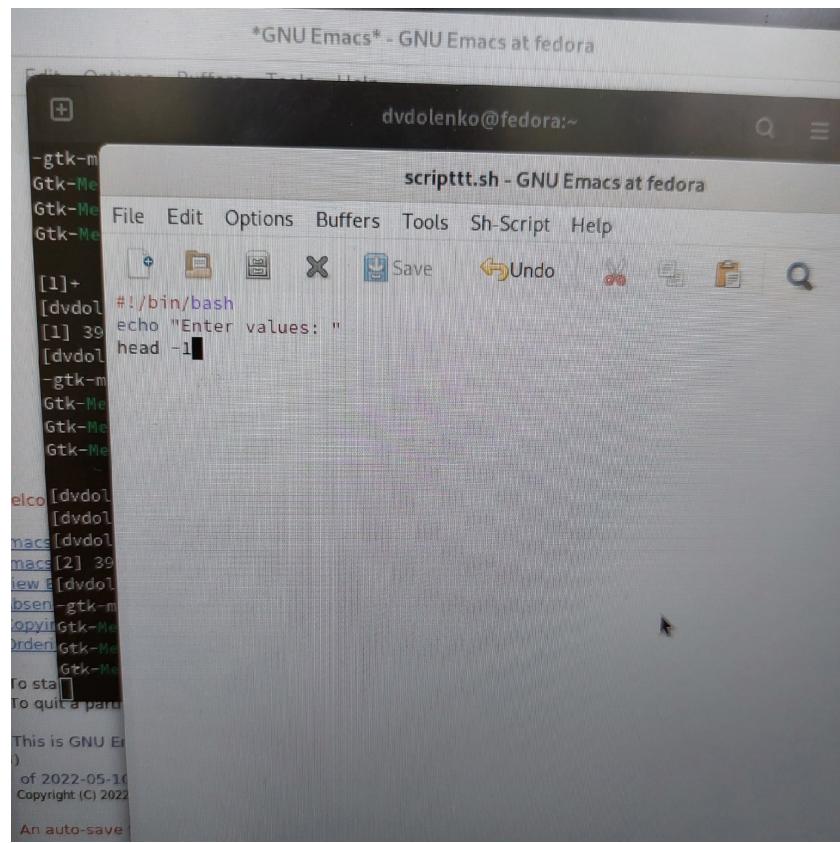
```
tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
```

```
tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
```

```
tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]
```

Manual page tar(1) line 1 (press h for help or q to quit).

Рис. 2.2: справка по архиватору



GNU Emacs - GNU Emacs at fedora

dvdolenko@fedora:~

scripttt.sh - GNU Emacs at fedora

File Edit Options Buffers Tools Sh-Script Help

```
[1]+  [dvdol] #!/bin/bash
[1] 39 echo "Enter values: "
[dvdol]
-gtk-m
Gtk-Me
Gtk-Me
Gtk-Me
elco [dvdol
[dvdol
macs [dvdol
macs [2] 39
new E [dvdol
bsen-gtk-m
copyingGtk-Me
Order Gtk-Me
Gtk-Me
To sta
To quit a part
This is GNU Em
)
of 2022-05-10
Copyright (C) 2022
An auto-save
```

Рис. 2.3: скрипт



Рис. 2.4: запускаю

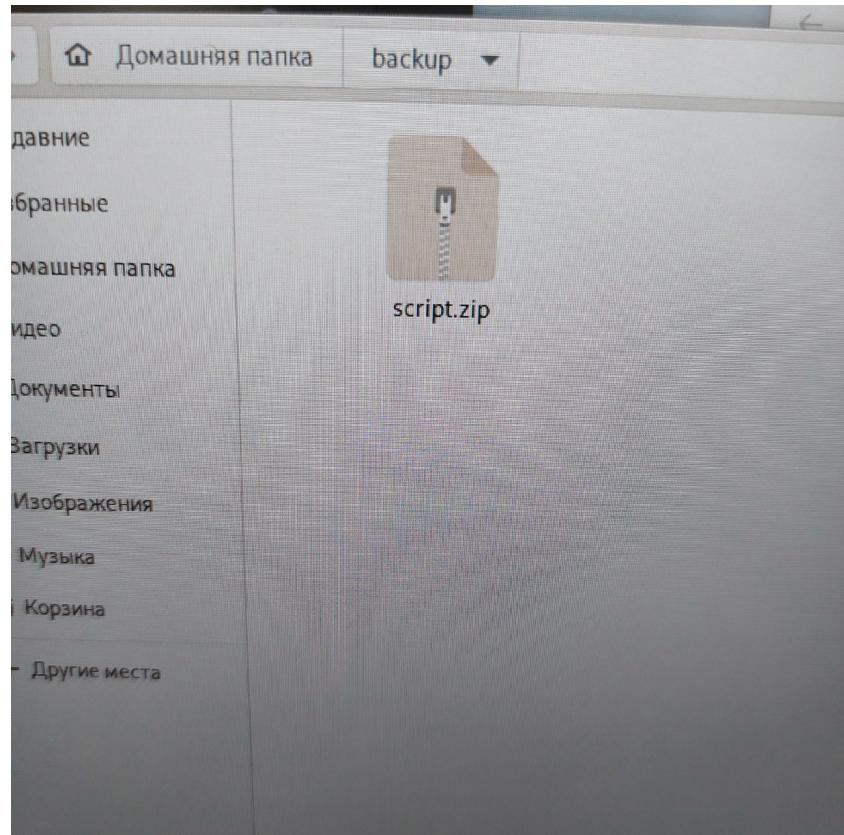
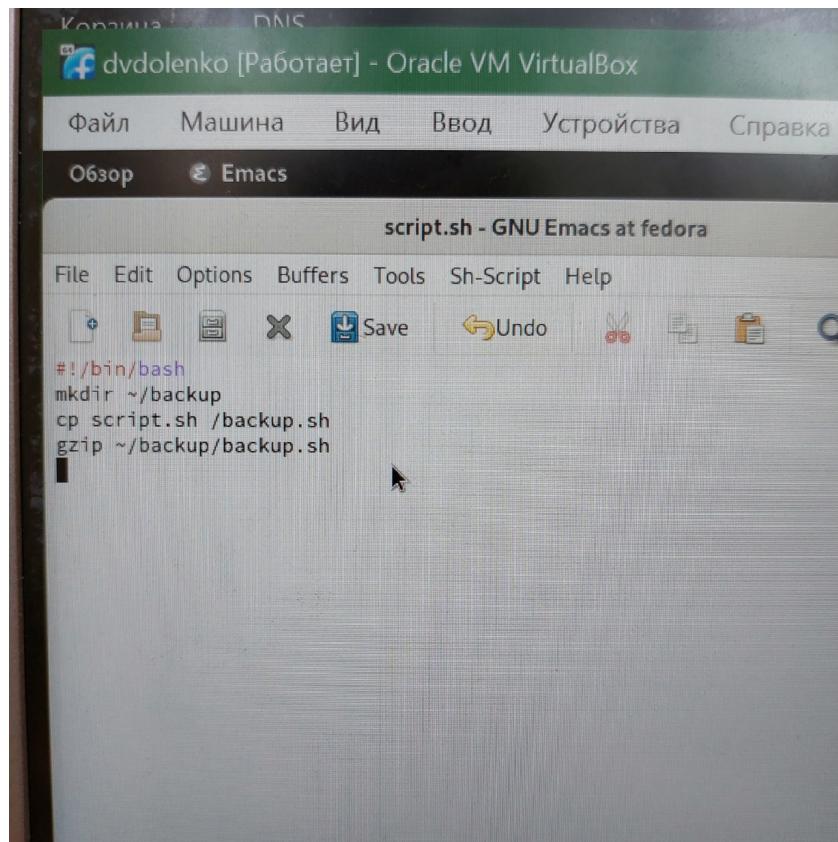


Рис. 2.5: создан архив

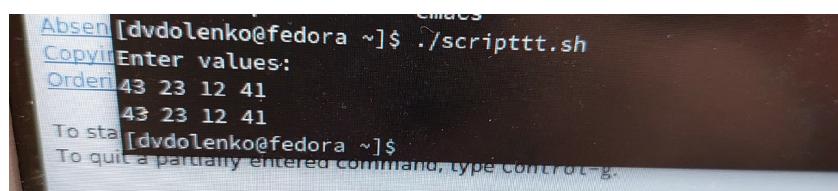
Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт последовательно распечатывает значения всех переданных аргументов. (рис. [-fig. 2.6]2.7)



The screenshot shows a window titled "dvdolenko [Работает] - Oracle VM VirtualBox". Inside, an Emacs window is open with the title "script.sh - GNU Emacs at fedora". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Below the menu is a toolbar with icons for file operations like Open, Save, Undo, and Redo. The main buffer contains the following shell script:

```
#!/bin/bash
mkdir ~/backup
cp script.sh /backup.sh
gzip ~/backup/backup.sh
```

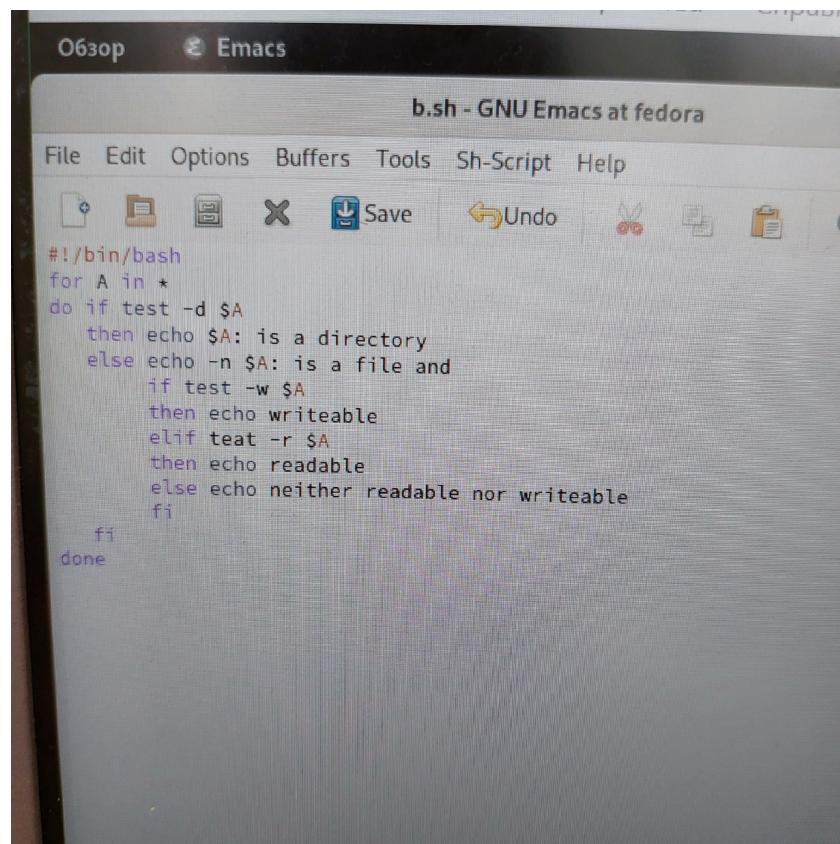
Рис. 2.6: скрипт



The screenshot shows a terminal window with the command `[dvdolenko@fedora ~]$./scripttt.sh` entered. The output shows the script running and printing the numbers 43, 23, 12, 41 twice. The terminal prompt `[dvdolenko@fedora ~]$` is visible at the bottom.

Рис. 2.7: запуск и проверка

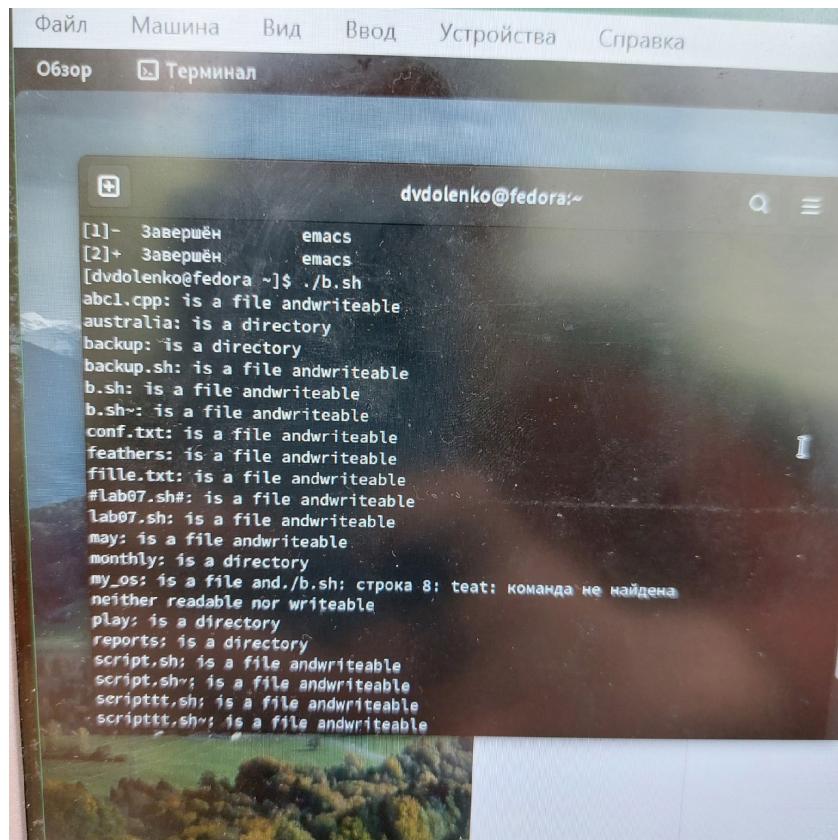
Написала командный файл—аналог команды ls.(рис. [-fig. 2.8]2.9)



The screenshot shows an Emacs window titled "b.sh - GNU Emacs at fedora". The window contains a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Below the menu is a toolbar with icons for file operations like Open, Save, Undo, and Redo. The main buffer area displays the following bash script:

```
#!/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 teat -r $A
        then echo readable
        else echo neither readable nor writeable
        fi
    done
```

Рис. 2.8: скрипт

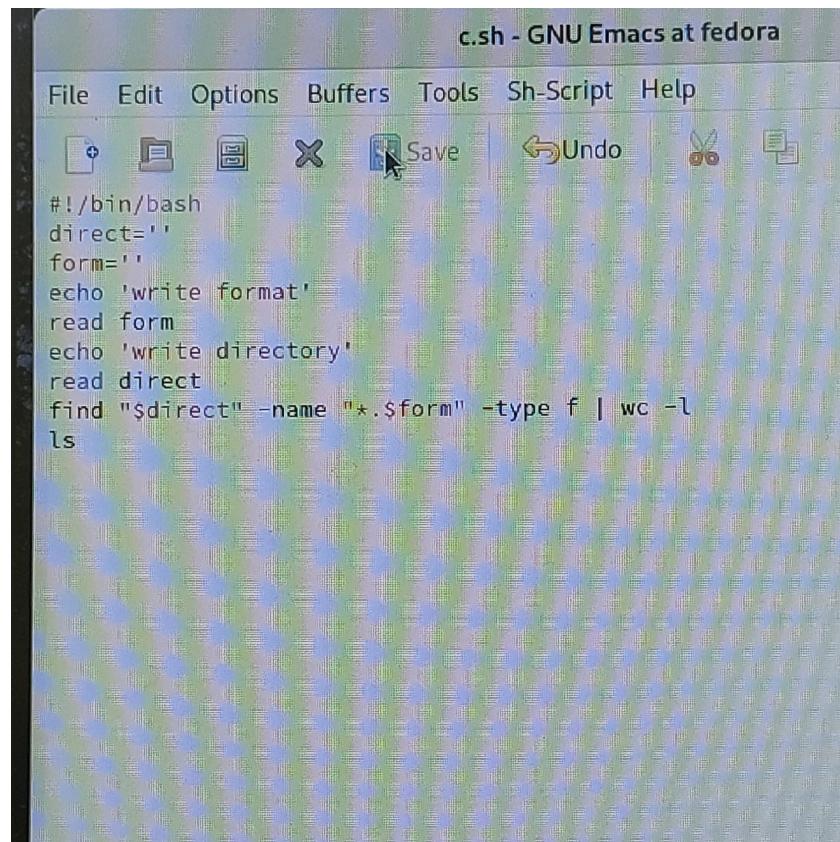


Файл Машина Вид Ввод Устройства Справка
Обзор Терминал

```
[1]- Завершён emacs
[2]+ Завершён emacs
[dvdolenko@fedora ~]$ ./b.sh
abc1.cpp: is a file andwriteable
australia: is a directory
backup: is a directory
backup.sh: is a file andwriteable
b.sh: is a file andwriteable
b.sh~: is a file andwriteable
conf.txt: is a file andwriteable
feathers: is a file andwriteable
fille.txt: is a file andwriteable
#lab07.sh#: is a file andwriteable
lab07.sh: is a file andwriteable
may: is a file andwriteable
monthly: is a directory
my_os: is a file and./b.sh: строка 8: teat: команда не найдена
neither readable nor writeable
play: is a directory
reports: is a directory
script.sh: is a file andwriteable
script.sh~: is a file andwriteable
scripttt.sh: is a file andwriteable
scripttt.sh~: is a file andwriteable
```

Рис. 2.9: запуск команды

Написала 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. [-fig. 2.10]2.11)



The screenshot shows a terminal window titled "c.sh - GNU Emacs at fedora". The window contains a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Below the menu is a toolbar with icons for file operations like Save and Undo. The main text area displays a bash script:

```
#!/bin/bash
direct=''
form=''
echo 'write format'
read form
echo 'write directory'
read direct
find "$direct" -name "*.$form" -type f | wc -l
ls
```

Рис. 2.10: скрипт

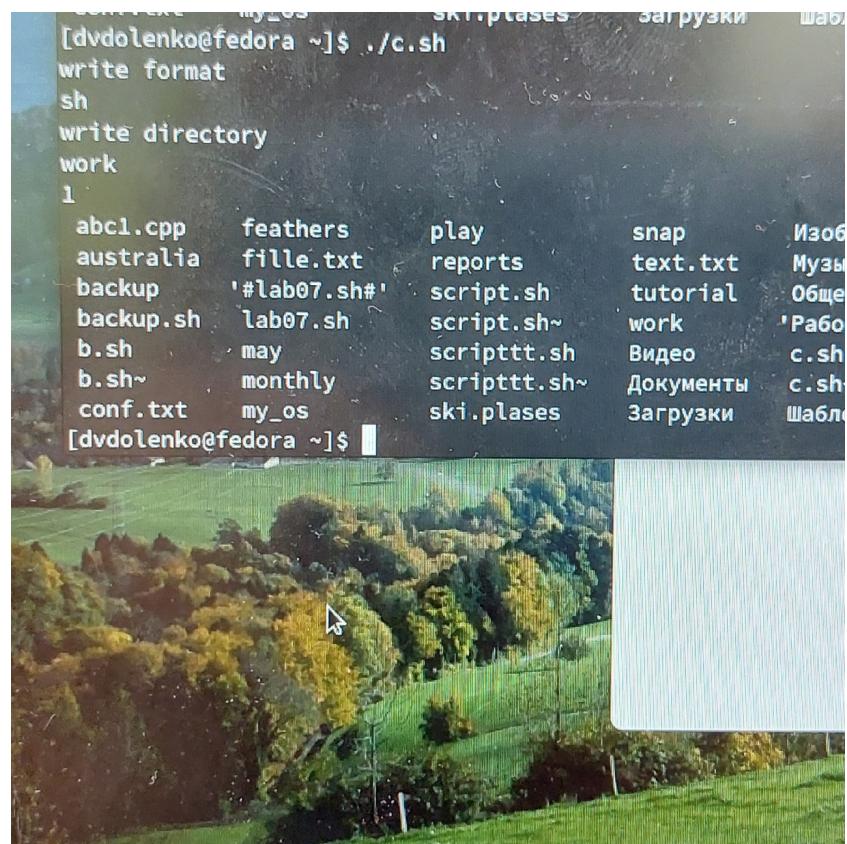


Рис. 2.11: запуск и проверка

3 Вывод

В ходе данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать небольшие командные файлы.

4 Ответы на 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;
 - С-оболочка (или csh) — надстройка над оболочкой Борна, использующая Сподобный 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 С 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/Linuxподобных 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 `$`. Например, команда `mv afile ${mark}` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}` Например, использование команд `b=/tmp/andyls -l myfile > blssudoapt –getinstalltexlive –luatexls/tmp/andy –ls, ls –l >bls` приведёт к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то её значением будет символ пробела. 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. 5. 6. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный. Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак

доллара; вы можете писать команды типа let sum=x+7, и let будет искать переменную x и добавлять к ней 7. Команда let также расширяет другие выражения let, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда let не ограничена простыми арифметическими выражениями. Команда read позволяет читать значения переменных со стандартного ввода: echo “Please enter Month and Day of Birth ?” read mon day trash В переменные mon и day будут считаны соответствующие значения, введённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.

5. – 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 (у Вас есть почта). – TERM – тип используемого терминала. – LOGNAME – содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему
6. 9. Такие символы, как ’ < > * ? | ” &, являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасим-

волов. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Стока, заключённая в двойные кавычки, экранирует все метасимволы, кроме \$, ', , “.

7. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: bash командный_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 +x имя_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.
8. Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом-f. Команда typeset имеет четыре опции для работы с функциями: -f — перечисляет определенные на текущий момент функции; --ft— при последующем вызове функции инициирует ее трассировку; --fx— экспортирует все перечисленные функции в любые дочерние программы оболочек; --fu— обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции.
9. ls -lrt Если есть d, то является файл каталогом
10. Для создания массива используется команда set с флагом -A. За флагом сле-

дует имя переменной, а затем список значений, разделённых пробелами. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `--f` — перечисляет определённые на текущий момент функции; `--ft` — при последующем вызове функции инициирует её трассировку; `--fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `--fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

11. 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 нее 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 нее имени данного командного файла. Рассмотрим это на примере. Пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1` Если Вы введете с терминала команду: `where andy`, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал не будет выведено ничего. Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандартный вывод. В этом

примере команда grep используется как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов andy, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации символов \$1 осуществляется подстановка значения первого и единственного параметра andy. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем andy, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: \$ where andy andy ttyG Jan 14 09:12 \$ Определим функцию, которая изменяет каталог и печатает список файлов: \$ function clist { > cd \$1 > ls > }. Теперь при вызове команды clist каталог будет изменен каталог и выведено его содержимое.

12. – \$* — 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.