

Отчет по лабораторной работе №11

Тема:

Программирование в командном процессоре ОС UNIX. Командные файлы

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Бен бадр Ясмин

Группа: НКНбд-01-20

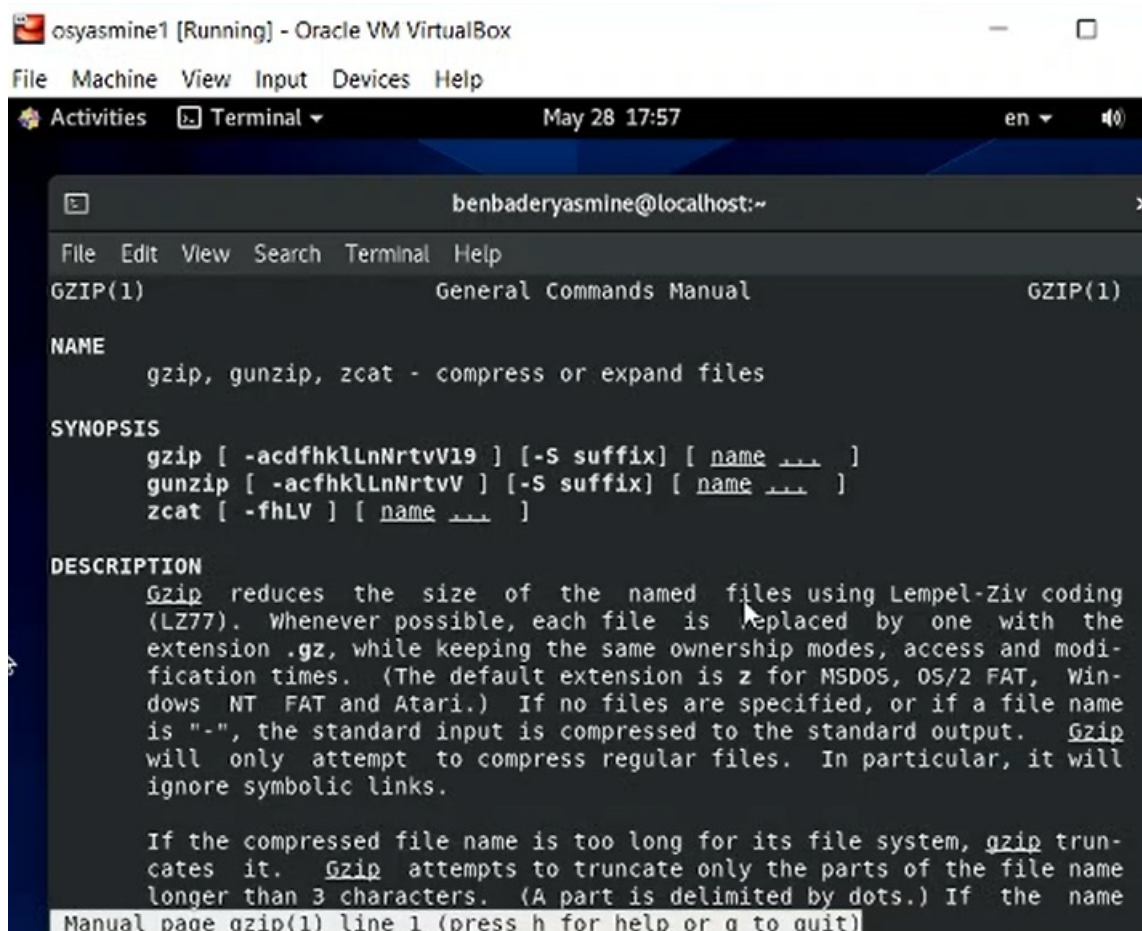
Москва, 2021г.

Цель работы

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

Ход работы:

1. сначала я вошла в gzip, Изучила опции команды, с помощью команды man.

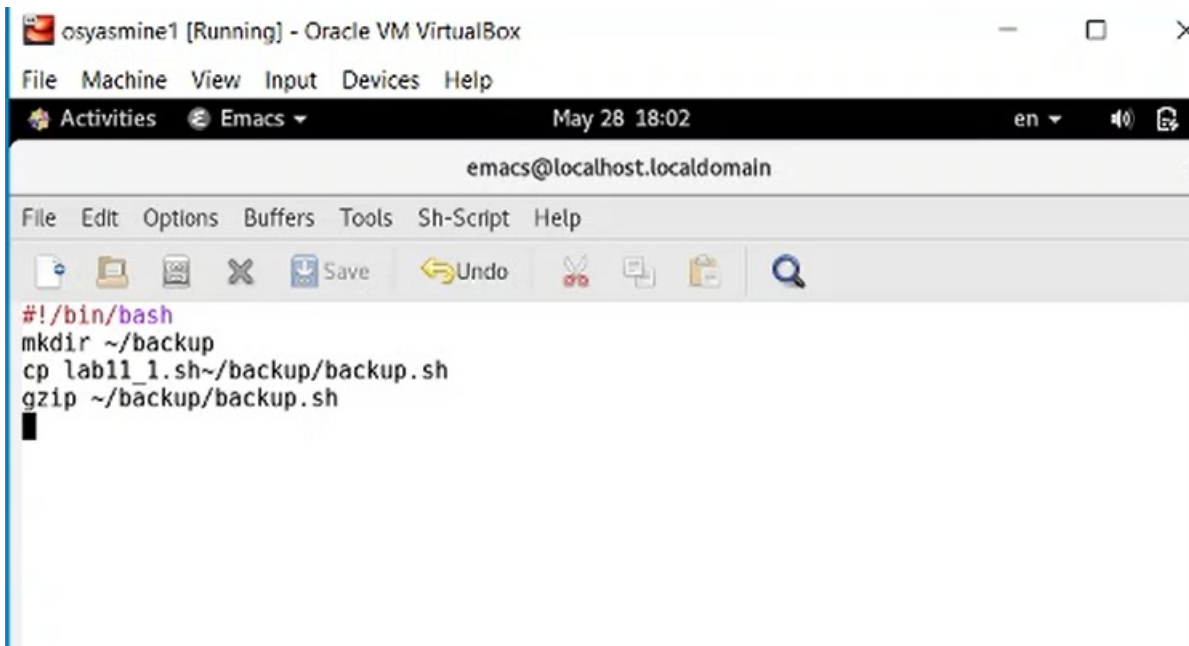


```
osyasmine1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal May 28 17:57 en
benbaderyasmine@localhost:~
File Edit View Search Terminal Help
GZIP(1) General Commands Manual GZIP(1)
NAME
gzip, gunzip, zcat - compress or expand files
SYNOPSIS
gzip [ -acdfhklLnRtvV19 ] [-S suffix] [ name ... ]
gunzip [ -acfhklLnRtvV ] [-S suffix] [ name ... ]
zcat [ -fhLV ] [ name ... ]
DESCRIPTION
Gzip reduces the size of the named files using Lempel-Ziv coding
(LZ77). Whenever possible, each file is replaced by one with the
extension .gz, while keeping the same ownership modes, access and modi-
fication times. (The default extension is z for MSDOS, OS/2 FAT, Win-
dows NT FAT and Atari.) If no files are specified, or if a file name
is "-", the standard input is compressed to the standard output. Gzip
will only attempt to compress regular files. In particular, it will
ignore symbolic links.
If the compressed file name is too long for its file system, gzip trun-
cates it. Gzip attempts to truncate only the parts of the file name
longer than 3 characters. (A part is delimited by dots.) If the name
Manual page gzip(1) line 1 (press h for help or q to quit)
```

- используя команду touch я Создала текстовый файл lab11_1.sh., Открывала текстовый редактор emacs.

```
benbaderyasmine@localhost:~  
File Edit View Search Terminal Help  
[benbaderyasmine@localhost ~]$ man gzip  
[benbaderyasmine@localhost ~]$ touch lab11_1.sh  
[benbaderyasmine@localhost ~]$ emacs
```

- Написала код, в котором я создаю папку backup, копирую текстовый файл lab11_1.sh, указав путь для сохранения файла. После архивировала данный файл через команду gzip.



osyasmine1 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Emacs May 28 18:02 en

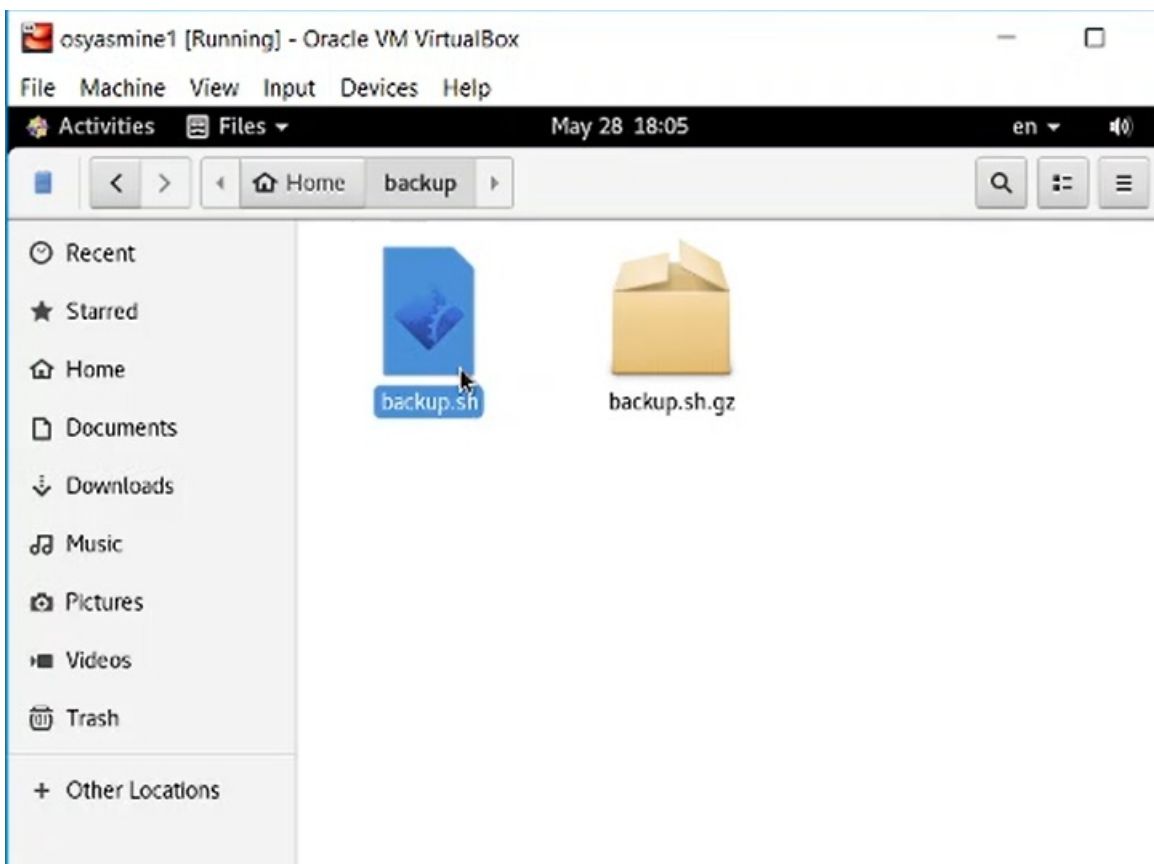
emacs@localhost.localdomain

File Edit Options Buffers Tools Sh-Script Help

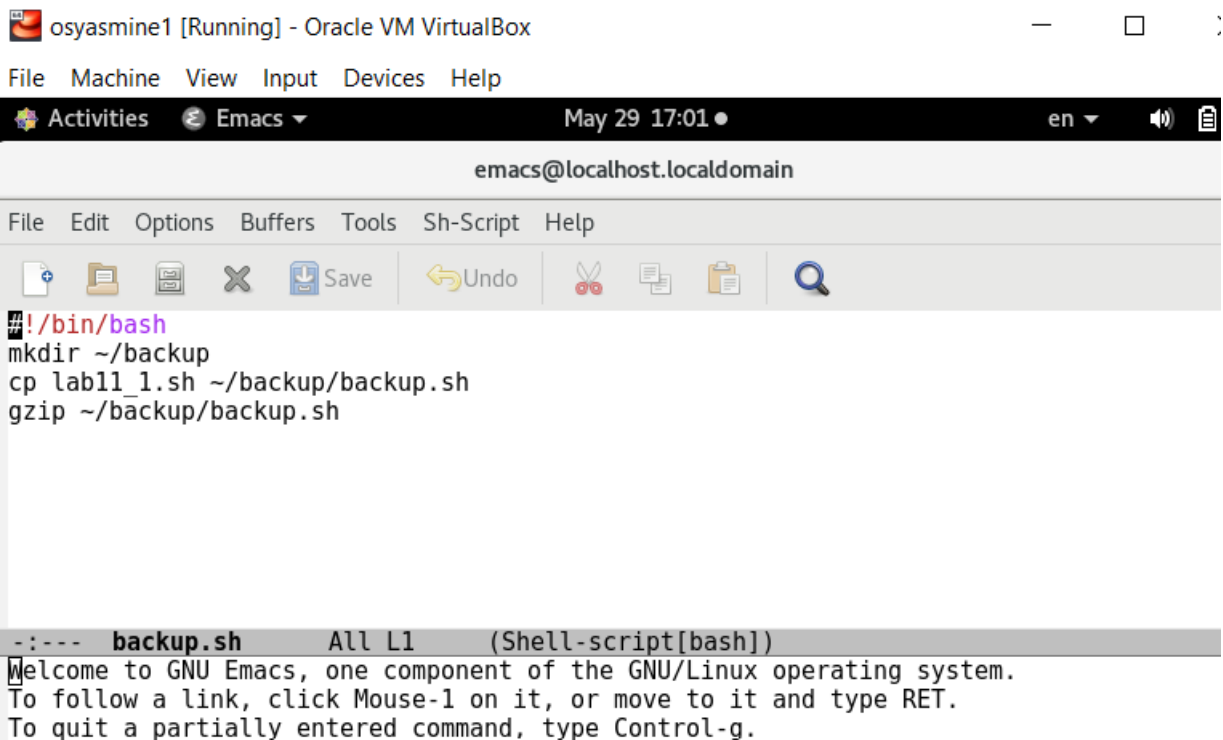
Save Undo

```
#!/bin/bash  
mkdir ~/backup  
cp lab11_1.sh~/backup/backup.sh  
gzip ~/backup/backup.sh
```

- написала Командой chmod чтобы файл был исполняемым в Linux. Проверила, если можно выполнить программу. Папка backup была создана.



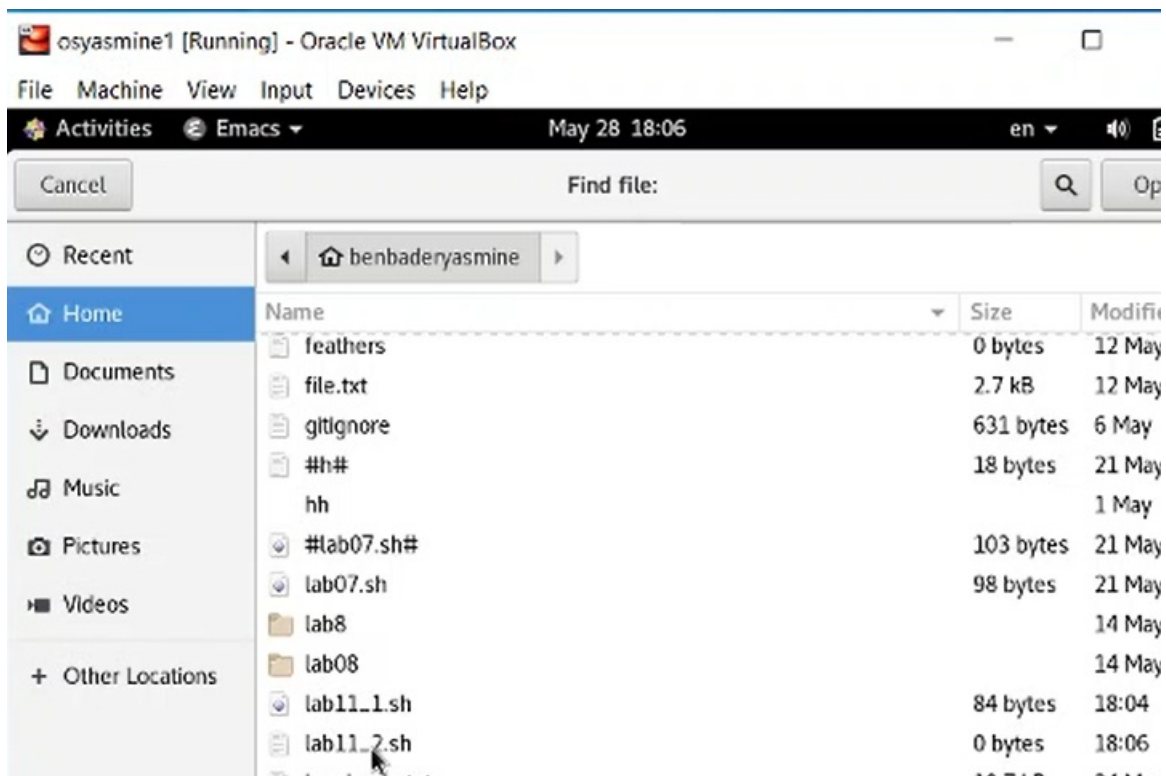
- Проверила, возможность выполнить мою программу.



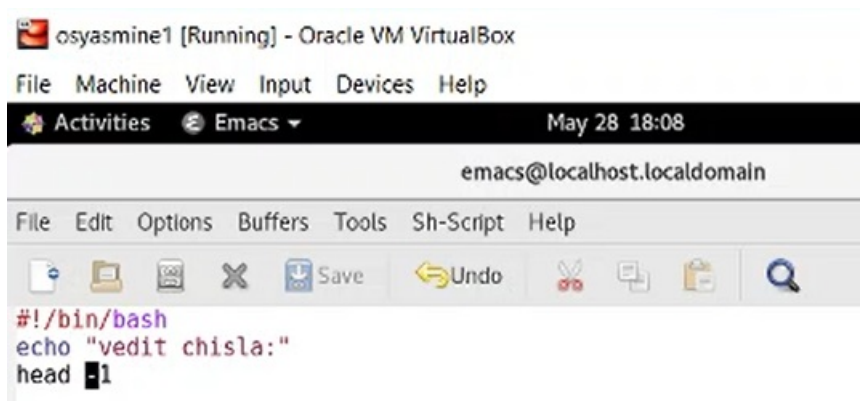
```
osyasmine1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Emacs May 29 17:01 en
emacs@localhost.localdomain
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
mkdir ~/backup
cp lab11_1.sh ~/backup/backup.sh
gzip ~/backup/backup.sh

---- backup.sh All L1 (Shell-script[bash])
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.
```

1. Создала новый файл lab11_2.sh, используя команду touch. Открываю текстовый редактор emacs.



- Написала программу, для вывода того, что я буду вводить.



```
osyasmine1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Emacs May 28 18:08
emacs@localhost.localdomain
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
echo "vedit chisla:"
head 1
```

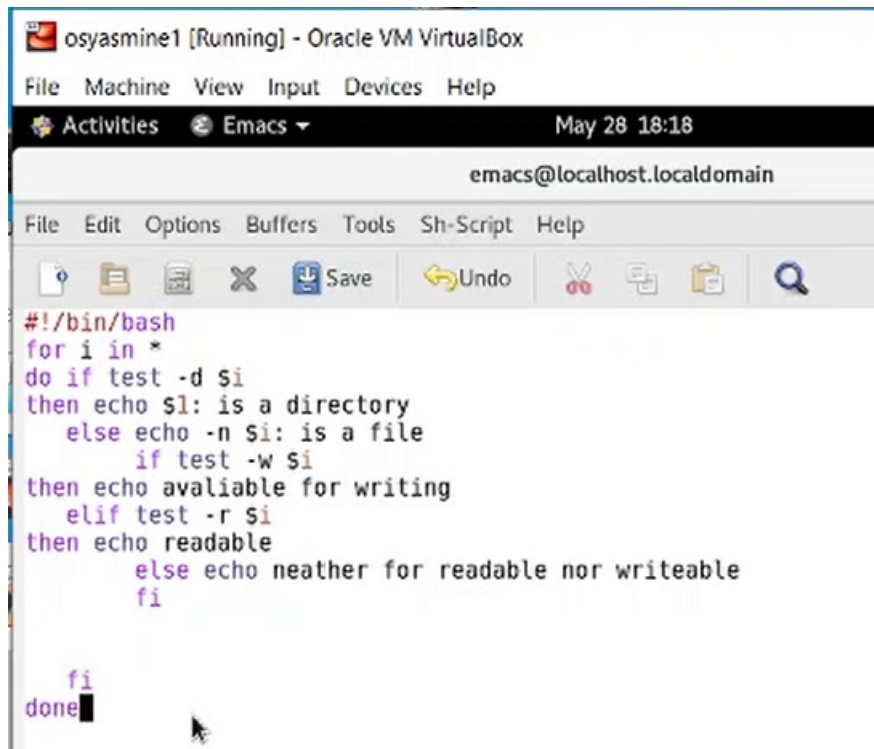
*использовала Командой chmod, чтобы файл был исполняемым в Linux. Следующая строка для того, чтобы он выполнил ранее написанную программу.

```
[benbaderyasmine@localhost ~]$ emacs
[benbaderyasmine@localhost ~]$ chmod +x lab11_2.sh
[benbaderyasmine@localhost ~]$ ./lab11_2.sh
edit chisla:
9 11 22 33 44 18
9 11 22 33 44 18
[benbaderyasmine@localhost ~]$
```

1. Создала текстовый файл lab11_3.sh, используя команду touch. Открываю текстовый редактор emacs.

```
9 11 22 33 44 18
[benbaderyasmine@localhost ~]$ touch lab11_3.sh
[benbaderyasmine@localhost ~]$ emacs
```

- Написала командный файл — аналог команды ls .



The screenshot shows the Emacs editor interface. The title bar reads "osyasmine1 [Running] - Oracle VM VirtualBox". The menu bar includes "File", "Machine", "View", "Input", "Devices", and "Help". The status bar at the top shows "May 28 18:18". The editor window title is "emacs@localhost.localdomain". The menu bar inside the editor includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". The toolbar contains icons for opening, saving, undo, redo, and search. The script content is as follows:

```
#!/bin/bash
for i in *
do if test -d $i
then echo $i: is a directory
else echo -n $i: is a file
if test -w $i
then echo available for writing
elif test -r $i
then echo readable
else echo neather for readable nor writeable
fi
fi
done
```

- выдавление информации о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

```
[benbaderyasmine@localhost ~]$ emacs
[benbaderyasmine@localhost ~]$ chmod +x lab11_3.sh
[benbaderyasmine@localhost ~]$ ./lab11_3.sh
```

```

ll.sh: is a fileavailable for writing
may: is a fileavailable for writing
: is a directory
: is a directory
my_os: is a fileavailable for writing
./lab11_3.sh: line 3: test: #new: binary operator expected
#new file#: is a file./lab11_3.sh: line 6: test: #new: binary operator expected
./lab11_3.sh: line 8: test: #new: binary operator expected
neather for readable nor writeable
: is a directory
play: is a fileavailable for writing
: is a directory
READ.md: is a fileavailable for writing
README.md: is a fileavailable for writing
: is a directory
: is a directory
: is a directory
: is a directory
: is a directory
: is a directory
: is a directory
: is a directory
: is a directory
text1.txt: is a fileavailable for writing
text.txt: is a fileavailable for writing
#tttttttttttttttttttttttttttttttttttt#: is a fileavailable for writing
: is a directory
: is a directory

```

1. Создала текстовый файл lab11_4.sh, используя команду touch. Открываю текстовый редактор emacs.

```

benbaderyasmine@localhost ~]$ touch lab11_4.sh
benbaderyasmine@localhost ~]$ emacs

```

- Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, pdf и т.д.).

```

#!/bin/bash
format="1"
direct="2"
echo "write format"
read format
echo "write direct"
read direct
find "$direct" -name ".sformat" -type f| wc -l
ls

```

- Вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.


```
benbaderyasmine@localhost ~/$ chmod +x lab11_4.sh
benbaderyasmine@localhost ~/$ ./lab11_4.sh
write format
sh
write direct
home/benbaderyasmine
find: 'home/benbaderyasmine': No such file or directory
9
abcl      '#h#'          legalcode.txt    ski.places
abcdl     hh            '#ll#'           Templates
amin      '#lab07.sh#'       ll.sh           test
amintest  lab07.sh          may             test1
australia lab08          monthly        test11
backup    lab11_1.sh         Music           test2
cc.txt     lab11_1.sh~        my_os           test3
conf.txt   lab11_2.sh         '#new file#'    text1.txt
Desktop    lab11_2.sh~        Pictures        text.txt
Documents  lab11_3.sh         play            '#####'
Downloads  lab11_3.sh~        Public          Videos
feathers   lab11_4.sh         READ.md         work
file.txt   lab11_4.sh~        README.md
gitignore  lab8              reports
benbaderyasmine@localhost ~/$
```

Вывод

Изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать небольшие командные файлы.

Бибблиография

Командные файлы Linux Командные процессоры (оболочки)

Ответы на контрольные вопросы:

1. Командные процессоры или оболочки – это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам. UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки:

* оболочка Борна (Bourne) – первоначальная командная оболочка UNIX: базовый, но полный набор функций; * C-оболочка – добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд; * оболочка Корна – 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/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 > ${b}lsudo apt-get install texlive-luatex` приведёт к переназначению стандартного вывода команды `ls` с терминала на файл `/tmp/andy-ls`, а использование команды `ls -l>${b}ls` 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 единственный 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. 7. — `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 (y Вас есть почта)`. — `TERM` — тип используемого терминала. — `LOGNAME` — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему 8. 9. Такие символы, как `<` `>` `*` `?` `|` `\` и являются метасимволами и имеют для командного процессора специальные

смысл. 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 `$`, `'`, `\`, `"`. 10. 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. 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`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определённые на текущий момент функции; `-ft` — при последующем вызове функции иницирует её трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочки; `-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. 12. `ls -lrt` Если есть `d`, то является файл каталогом 13. Для создания массива используется команда `set` с флагом `-A`. За флагом следует 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 иницирует её трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочки; `-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. 14. 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 введёте 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. 15. `- $*` — 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:~}` — 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`.