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: Мухамедияр Адиль

Группа: НКНбд-01-20

Москва, 2021г.

Цель работы

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

Введение

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:

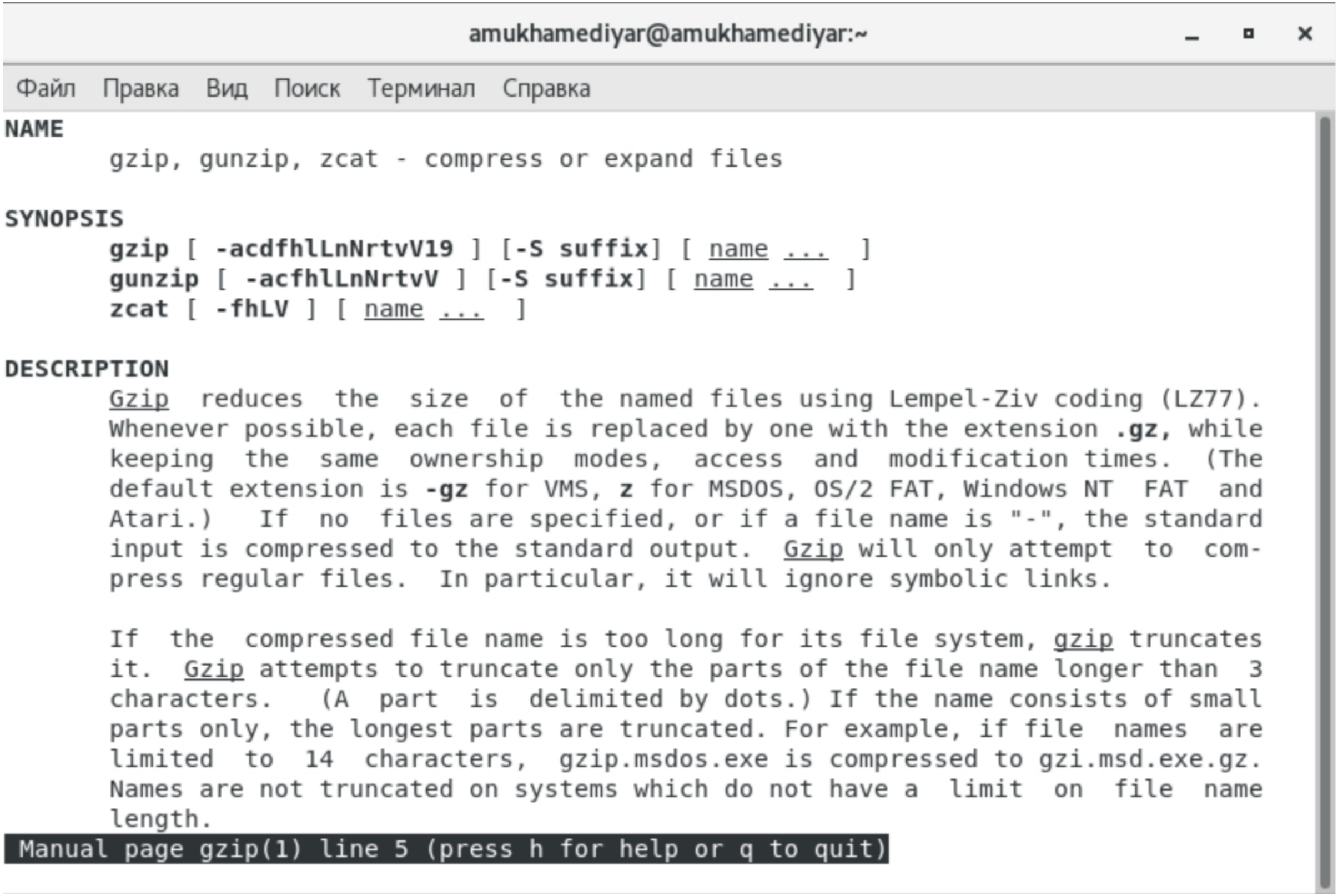
- оболочка Борна (Bourne) – первоначальная командная оболочка UNIX: базовый, но полный набор функций;
- C-оболочка – добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд;
- оболочка Корна – 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).

Последовательность выполнения работы

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

Ход работы:

1. Изучил опции команды, и саму команду *gzip*, с помощью команды *man*.



- Создал текстовой файл *lab11.sh*, используя команду *touch*. Открываю текстовой редактор *emacs*.

```
[amukhamediyar@amukhamediyar ~]$ man gzip
[amukhamediyar@amukhamediyar ~]$ touch lab11.sh
[amukhamediyar@amukhamediyar ~]$ emacs
```

- Написал код, в котором я создаю папку *backup*, копировал текстовой файл *lab11.sh*, указав путь для сохранения файла. После архивировал данный файл через команду *gzip*.

lab11.sh - emacs@amukhamediyar.localdomain

File Edit Options Buffers Tools Sh-Script Help

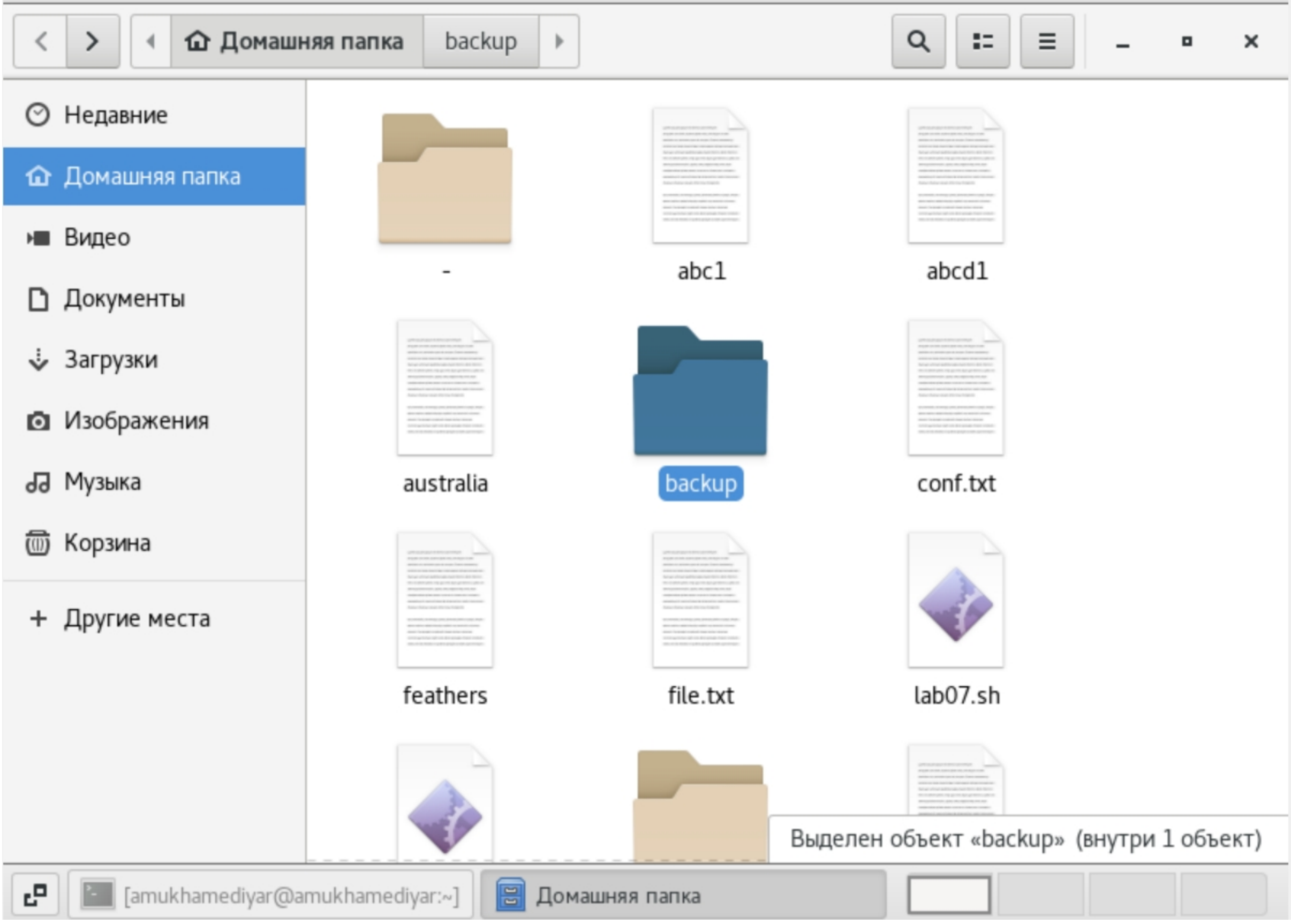
Save Undo

```
#!/bin/bash
mkdir ~/backup
cp lab11.sh ~/backup/backup.sh
gzip ~/backup/backup.sh
```

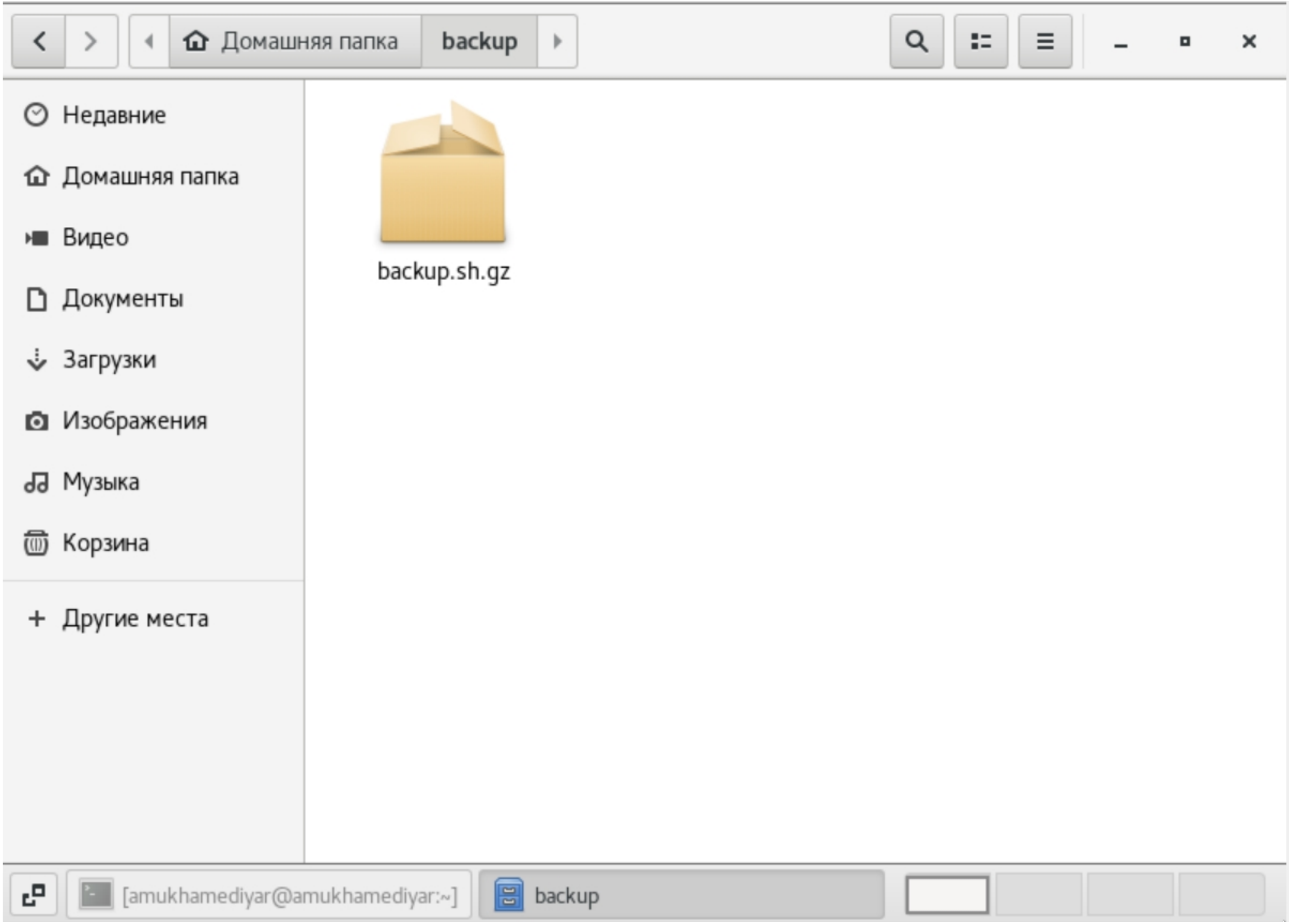
- Командой *chomad* я сделал так, чтобы файл был исполняемым в **Linux**. Следующая строка для того, чтобы он выполнил нашу ранее написанную программу.

```
[amukhamediyar@amukhamediyar ~]$ chmod +x lab11.sh
[amukhamediyar@amukhamediyar ~]$ ./lab11.sh
[amukhamediyar@amukhamediyar ~]$
```

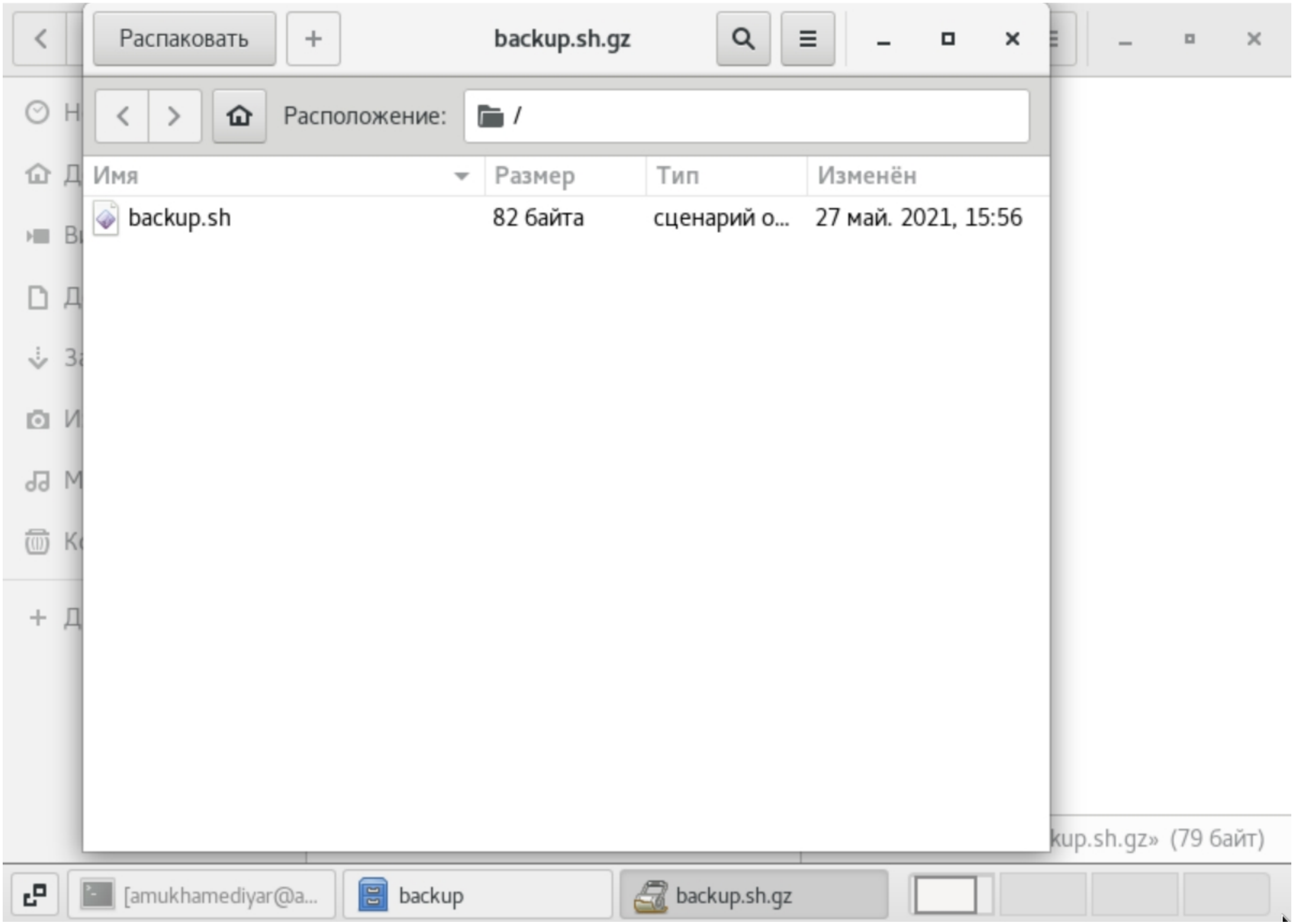
- Проверяю, смог ли выполнить мою программу. Папка с именем *backup* была создана. Видим, папка с именем *backup.sh* был архивирован, теперь называется *backup.sh.gz*.



- Проверяю, смог ли выполнить мою программу.



- Проверяю, смог ли выполнить мою программу. Внутри папки *backup.sh.gz* находиться наш скопированный файл *lab11.sh*, который был переименован в *backup.sh*.



2. Создал текстовой файл *lab11_2.sh*(для выполнения 2 пункта данной лабораторной работы), используя команду *touch*. Открываю текстовой редактор *emacs*.

```
[amukhamediyar@amukhamediyar ~]$ touch lab11_2.sh
[amukhamediyar@amukhamediyar ~]$ emacs
```

- Написал программу, для вывода того, что я буду вводить. Для начало надо ввести числа, я решил ввести нечетные числа, как видим, выводит так же нечетные, при это те же самые, которые были введены.

lab11_2.sh - emacs@amukhamediyar.localdomain

File Edit Options Buffers Tools Sh-Script Help

Save Undo

```
#!/bin/bash
echo "Vvedite chisla:"
head -1
```

- Командой *chmad* я сделал так, чтобы файл был исполняемым в **Linux**. Следующая строка для того, чтобы он выполнил нашу ранее написанную программу.

```
[amukhamediyar@amukhamediyar ~]$ chmod +x lab11_2.sh
[amukhamediyar@amukhamediyar ~]$ ./lab11_2.sh
Vvedite chisla:
1 3 5 7 9 11
1 3 5 7 9 11
[amukhamediyar@amukhamediyar ~]$
```

3. Создал текстовой файл *lab11_3.sh*(для выполнения 2 пункта данной лабораторной работы), используя команду *touch*. Открываю текстовой редактор *emacs*.

```
[amukhamediyar@amukhamediyar ~]$ touch lab11_3.sh
[amukhamediyar@amukhamediyar ~]$ emacs
```

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

lab11_3.sh - emacs@amukhamediyar.localdomain

File Edit Options Buffers Tools Sh-Script Help

Save Undo

```
#!/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 avaliable for writing
        elif test -r $i
        then echo readable
        else echo neather for readable nor writeable
        fi
   fi
done
```

- Он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

```
[amukhamediyar@amukhamediyar ~]$ chmod +x lab11_3.sh
[amukhamediyar@amukhamediyar ~]$ ./lab11_3.sh
-: is a directory
abc1: is a fileavaliable for writing
abcd1: is a fileavaliable for writing
australia: is a fileavaliable for writing
backup: is a directory
conf.txt: is a fileavaliable for writing
feathers: is a fileavaliable for writing
file.txt: is a fileavaliable for writing
#lab#: is a fileavaliable for writing
#lab07.sh#: is a fileavaliable for writing
lab07.sh: is a fileavaliable for writing
lab_08: is a directory
lab11_2.sh: is a fileavaliable for writing
lab11_2.sh~: is a fileavaliable for writing
lab11_3.sh: is a fileavaliable for writing
lab11_3.sh~: is a fileavaliable for writing
lab11.sh: is a fileavaliable for writing
lab11.sh~: is a fileavaliable for writing
may: is a fileavaliable for writing
monthly: is a directory
my_os: is a fileavaliable for writing
play: is a fileavaliable for writing
reports: is a directory
ski.places: is a directory
```

4. Создал текстовой файл *lab11_4.sh*(для выполнения 2 пункта данной лабораторной работы), используя команду *touch*. Открываю текстовой редактор *emacs*.

```
[amukhamediyar@amukhamediyar ~]$ touch lab11_4.sh
[amukhamediyar@amukhamediyar ~]$ emacs
```

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

lab11_4.sh - emacs@amukhamediyar.localdomain

File Edit Options Buffers Tools Sh-Script Help

Save Undo

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

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

```
[amukhamediyar@amukhamediyar ~]$ chmod +x lab11_4.sh
[amukhamediyar@amukhamediyar ~]$ ./lab11_4.sh
write format
.sh
write direct
home/amukhamediyar
find: ‘home/amukhamediyar’: Нет такого файла или каталога
find: ‘name’: Нет такого файла или каталога
find: ‘..sh’: Нет такого файла или каталога
28
-          feathers    lab11_2.sh   lab11.sh   reports     work          Общедоступные
abc1       file.txt    lab11_2.sh~  lab11.sh~  ski.places  Видео         Рабочий стол
abcd1      #lab#       lab11_3.sh   may        test        Документы     Шаблоны
australia  #lab07.sh#  lab11_3.sh~  monthly    testlab     Загрузки
backup     lab07.sh    lab11_4.sh   my_os      text.txt    Изображения
conf.txt   lab_08      lab11_4.sh~  play       tutorial    Музыка
[amukhamediyar@amukhamediyar ~]$
```

Вывод

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

Библиография

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;
- С-оболочка – добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя С-подобный синтаксис команд, и сохраняет историю выполненных команд;
- оболочка Корна – напоминает оболочку С, 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 > ${b}lssudo apt-get install texlive-luatex приведёт к переназначению стандартного вывода команды ls с терминала на файл /tmp/andy-ls, а использование команды ls -l>$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. 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 (у Вас есть почта). – TERM — тип используемого терминала. – LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему 8, 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 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, 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. 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 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. 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 команд- ном 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. 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: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.