

Лабораторная работа №10

НКАбд-01-22

Никита Михайлович Демидович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Контрольные вопросы	21
6	Выводы	31
	Список литературы	32

Список иллюстраций

4.1	Создание исполняемого файла programm1.sh	9
4.2	Исполняемый файл programm1.sh	10
4.3	Создание директории backup	11
4.4	Работа исполняемого файла programm1.sh	11
4.5	Создание исполняемого файла programm2.sh	12
4.6	Исполняемый файл programm2.sh	13
4.7	Работа исполняемого файла programm2.sh	14
4.8	Создание исполняемого файла programm3.sh	15
4.9	Исполняемый файл programm3.sh	16
4.10	Работа исполняемого файла programm3.sh	17
4.11	Создание исполняемого файла programm4.sh	18
4.12	Исполняемый файл programm3.sh	19
4.13	Работа исполняемого файла programm4.sh	20

Список таблиц

1 Цель работы

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

2 Задание

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.
4. Написать 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.

3 Теоретическое введение

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 C-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).

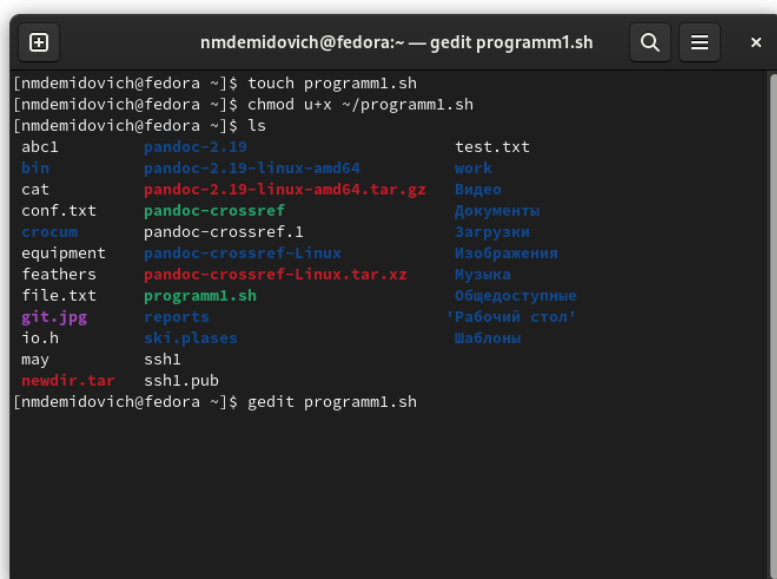
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на

базе оболочки Korn. Рассмотрим основные элементы программирования в оболочке `bash` (в разделе “Выполнение лабораторной работы”). В других оболочках большинство команд будет совпадать с описанными ниже.

4 Выполнение лабораторной работы

На первом этапе выполнения работы я создал командный файл `programm1.sh` и приступил к написанию скрипта, который при запуске делает резервную копию самого себя в директорию `backup` в домашнем каталоге (рис. [4.1]) - (рис. [4.2]):

```
tar -cvf ~/backup/backup.tar programm1.sh
```



```
nmdemidovich@fedora:~ — gedit programm1.sh
[nmdemidovich@fedora ~]$ touch programm1.sh
[nmdemidovich@fedora ~]$ chmod u+x ~/programm1.sh
[nmdemidovich@fedora ~]$ ls
abcl      pandoc-2.19      test.txt
bin       pandoc-2.19-linux-amd64  work
cat       pandoc-2.19-linux-amd64.tar.gz  Видео
conf.txt  pandoc-crossref   Документы
crocum    pandoc-crossref.1  Загрузки
equipment pandoc-crossref-Linux  Изображения
feathers   pandoc-crossref-Linux.tar.xz  Музыка
file.txt   programm1.sh      Общедоступные
git.jpg    reports           'Рабочий стол'
io.h       ski.plases        Шаблоны
may        sshl
newdir.tar sshl.pub
[nmdemidovich@fedora ~]$ gedit programm1.sh
```

Рис. 4.1: Создание исполняемого файла `programm1.sh`

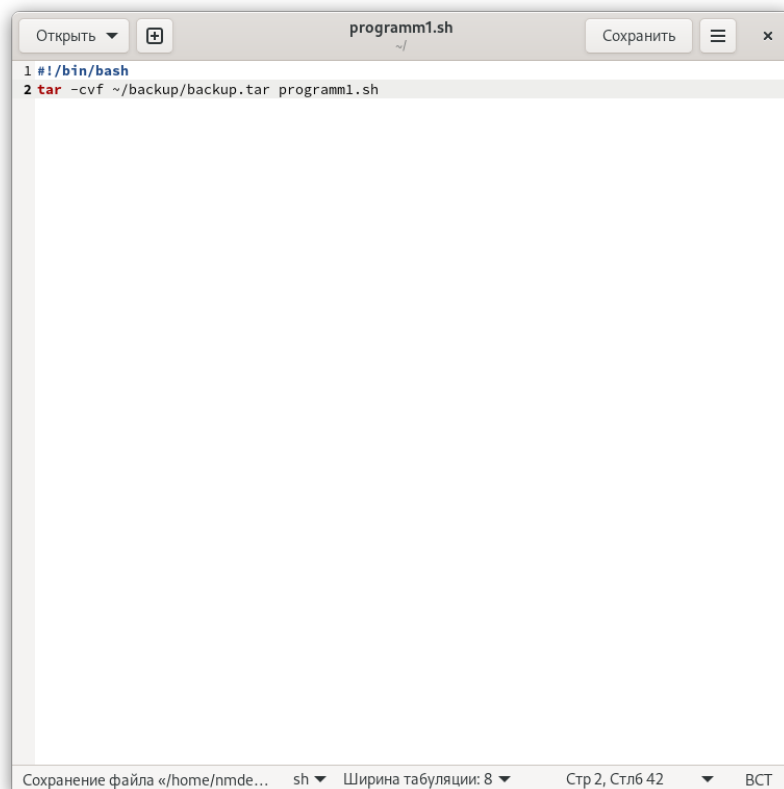


Рис. 4.2: Исполняемый файл programm1.sh

Затем я создал данную директорию, куда в последствии сохранялась бы резервная копия файла, после чего запустил написанный ранее скрипт и проверил его работу (рис. [4.3]) - (рис. [4.4]).

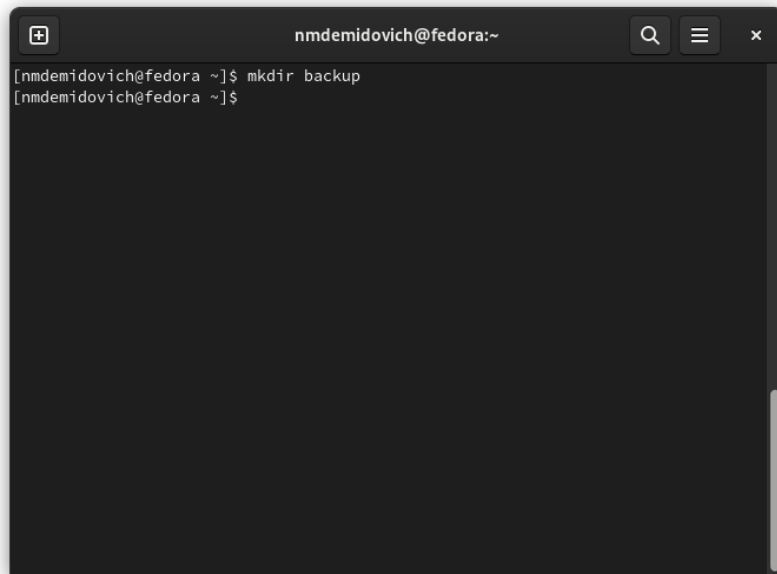
A terminal window titled 'nmdemidovich@fedora:~' with search, menu, and close icons. It shows the command 'mkdir backup' being executed successfully, with the prompt returning to '[nmdemidovich@fedora ~]\$'.

Рис. 4.3: Создание директории backup

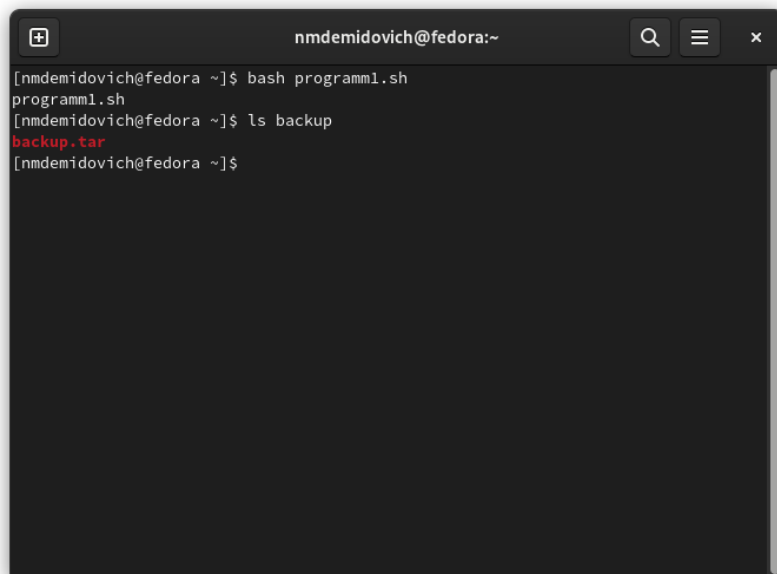
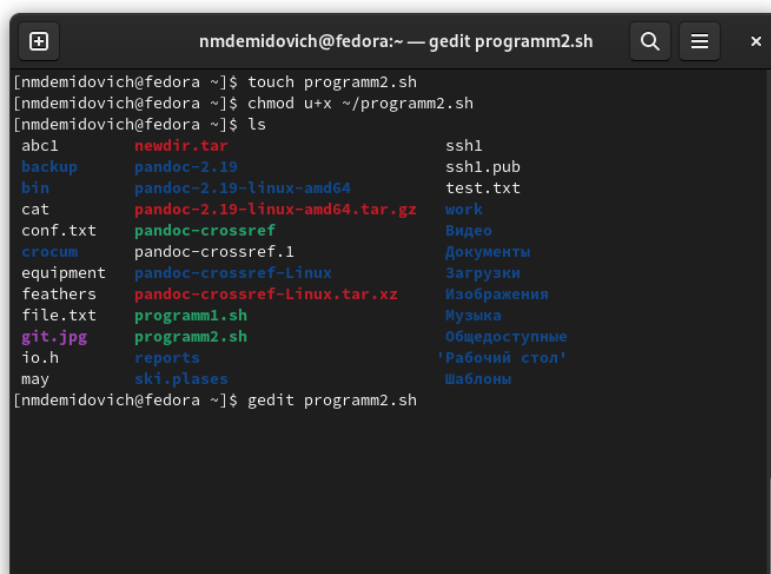
A terminal window titled 'nmdemidovich@fedora:~' with search, menu, and close icons. It shows the command 'bash program1.sh' being executed, which runs 'program1.sh'. Then, the command 'ls backup' is executed, showing the output 'backup.tar' in red text. The prompt returns to '[nmdemidovich@fedora ~]\$'.

Рис. 4.4: Работа исполняемого файла program1.sh

После этого я приступил к выполнению второго задания. Мною был написан командный файл `programm2.sh`, который обрабатывает любое произвольное

число аргументов командной строки, в том числе превышающее десять (10) (рис. [4.5]) - (рис. [4.7]):

```
# echo 'Введите любые числа'
# read n
for A in $*
do echo $A
done
```



The screenshot shows a terminal window titled "nmdemidovich@fedora:~ — gedit programm2.sh". The terminal output is as follows:

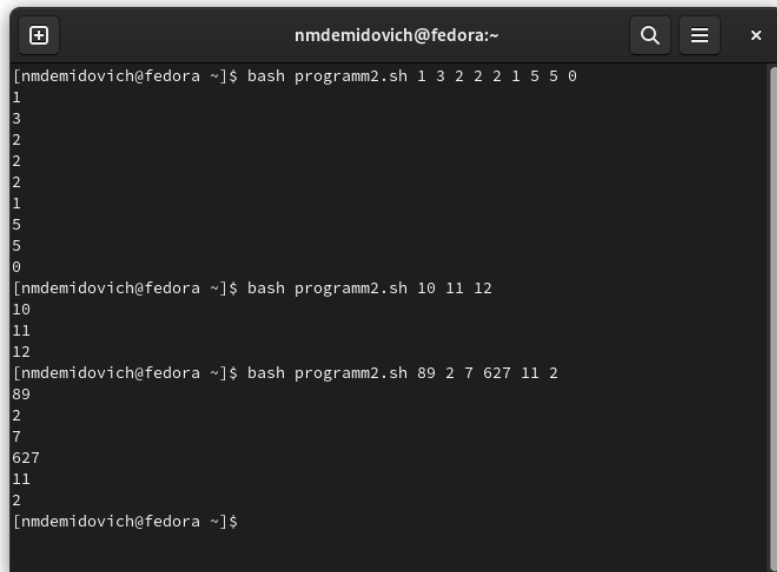
```
[nmdemidovich@fedora ~]$ touch programm2.sh
[nmdemidovich@fedora ~]$ chmod u+x ~/programm2.sh
[nmdemidovich@fedora ~]$ ls
abcl          newdir.tar          sshl
backup        pandoc-2.19          sshl.pub
bin           pandoc-2.19-linux-amd64  test.txt
cat           pandoc-2.19-linux-amd64.tar.gz  work
conf.txt      pandoc-crossref       Видео
crocum        pandoc-crossref.1     Документы
equipment     pandoc-crossref-Linux  Загрузки
feathers       pandoc-crossref-Linux.tar.xz  Изображения
file.txt      programm1.sh          Музыка
git.jpg       programm2.sh          Общедоступные
io.h          reports              'Рабочий стол'
may           ski.plases           Шаблоны
[nmdemidovich@fedora ~]$ gedit programm2.sh
```

Рис. 4.5: Создание исполняемого файла programm2.sh

```
1 #!/bin/bash
2 # echo 'Введите любые числа'
3 # read n
4 for A in $*
5 do echo $A
6 done
```

Сохранение файла «/home/nmde... sh ▾ Ширина табуляции: 8 ▾ Стр 6, Стлб 5 ▾ ВСТ

Рис. 4.6: Исполняемый файл programm2.sh



```
nmdemidovich@fedora:~  
[nmdemidovich@fedora ~]$ bash programm2.sh 1 3 2 2 2 1 5 5 0  
1  
3  
2  
2  
2  
1  
5  
5  
0  
[nmdemidovich@fedora ~]$ bash programm2.sh 10 11 12  
10  
11  
12  
[nmdemidovich@fedora ~]$ bash programm2.sh 89 2 7 627 11 2  
89  
2  
7  
627  
11  
2  
[nmdemidovich@fedora ~]$
```

Рис. 4.7: Работа исполняемого файла programm2.sh

Далее я написал командный файл programm3.sh, который является аналогом команды ls (без использования самой этой команды и команды dir): он выдает информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога (рис. [4.8]) - (рис. [4.10]):

```
for A in *  
do  
    if test -d "$A"  
    then  
        echo -n "$A: is a directory"  
    else  
        echo -n "$A: is a file and "  
        if test -w $A  
        then  
            echo writeable  
            if test -r $A
```

```

then
    echo "readable"
else
    echo "neither readable nor writeable"
fi
fi
fi
done

```

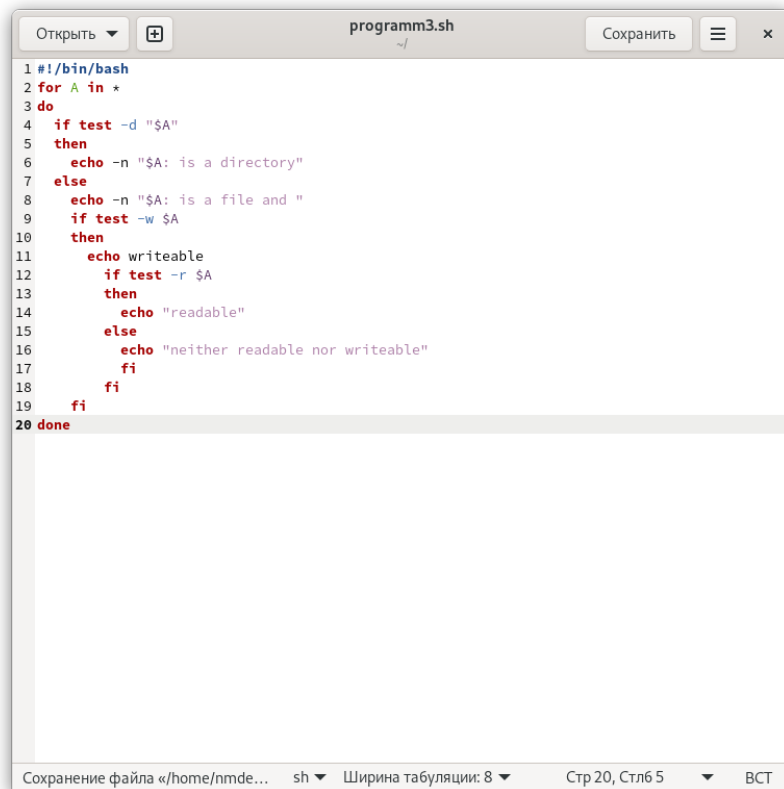
The screenshot shows a terminal window titled "nmdemidovich@fedora:~ — gedit programm3.sh". The terminal output is as follows:

```

[nmdemidovich@fedora ~]$ touch programm3.sh
[nmdemidovich@fedora ~]$ chmod u+x ~/programm3.sh
[nmdemidovich@fedora ~]$ ls
abcl      io.h      programm1.sh  Документы
backup    may       programm2.sh  Загрузки
bin        newdir.tar programm3.sh  Изображения
cat        pandoc-2.19 reports      Музыка
conf.txt   pandoc-2.19-linux-amd64 ski.plases   Общедоступные
crocum     pandoc-2.19-linux-amd64.tar.gz sshl         'Рабочий стол'
equipment  pandoc-crossref sshl.pub     Шаблоны
feathers    pandoc-crossref.1 test.txt
file.txt    pandoc-crossref-Linux work
git.jpg     pandoc-crossref-Linux.tar.xz video
[nmdemidovich@fedora ~]$ gedit programm3.sh

```

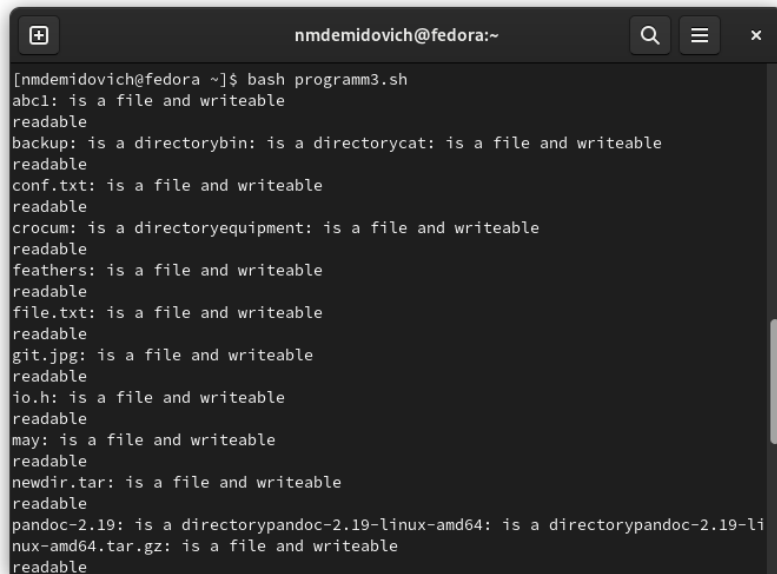
Рис. 4.8: Создание исполняемого файла programm3.sh



```
1 #!/bin/bash
2 for A in *
3 do
4     if test -d "$A"
5     then
6         echo -n "$A: is a directory"
7     else
8         echo -n "$A: is a file and "
9         if test -w $A
10        then
11            echo writeable
12            if test -r $A
13            then
14                echo "readable"
15            else
16                echo "neither readable nor writeable"
17            fi
18        fi
19    fi
20 done
```

Сохранение файла «/home/nmde... sh ▾ Ширина табуляции: 8 ▾ Стр 20, Стлб 5 ▾ ВСТ

Рис. 4.9: Исполняемый файл programm3.sh

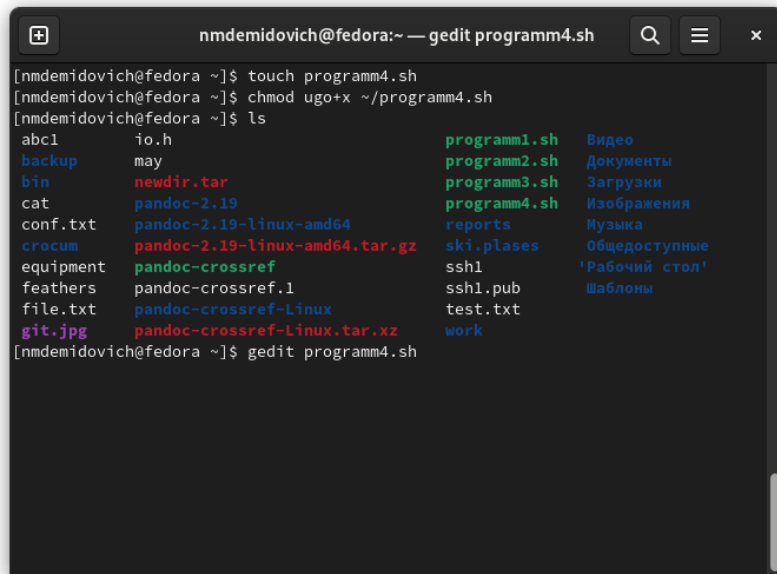
A terminal window titled 'nmdemidovich@fedora:~' with search, menu, and close icons. It shows the execution of a script 'programm3.sh'. The script iterates through a list of files and directories, checking their permissions and printing the results. The output shows that most files are readable and writeable, while some directories are only readable.

```
[nmdemidovich@fedora ~]$ bash programm3.sh
abcl: is a file and writeable
readable
backup: is a directorybin: is a directorycat: is a file and writeable
readable
conf.txt: is a file and writeable
readable
crocum: is a directoryequipment: is a file and writeable
readable
feathers: is a file and writeable
readable
file.txt: is a file and writeable
readable
git.jpg: is a file and writeable
readable
io.h: is a file and writeable
readable
may: is a file and writeable
readable
newdir.tar: is a file and writeable
readable
pandoc-2.19: is a directorypandoc-2.19-linux-amd64: is a directorypandoc-2.19-li
nux-amd64.tar.gz: is a file and writeable
readable
```

Рис. 4.10: Работа исполняемого файла programm3.sh

И на финальном этапе выполнения работы я создал командный файл programm4.sh, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории (рис. [4.11]) - (рис. [4.13]):

```
format=""
directory=""
echo "Введите формат файла: "
read format
echo "Введите директорию: "
read directory
find "${directory}" -name "*.${format}" -type f | wc -l
ls
```



```
nmdemidovich@fedora:~ — gedit programm4.sh
[nmdemidovich@fedora ~]$ touch programm4.sh
[nmdemidovich@fedora ~]$ chmod ugo+x ~/programm4.sh
[nmdemidovich@fedora ~]$ ls
abc1      io.h      programm1.sh  Видео
backup    may       programm2.sh  Документы
bin        newdir.tar programm3.sh  Загрузки
cat        pandoc-2.19  programm4.sh  Изображения
conf.txt   pandoc-2.19-linux-amd64  reports      Музыка
crocum     pandoc-2.19-linux-amd64.tar.gz  ski.places   Общедоступные
equipment  pandoc-crossref  ssh1         'Рабочий стол'
feathers    pandoc-crossref.1  ssh1.pub     Шаблоны
file.txt    pandoc-crossref-Linux  test.txt
git.jpg     pandoc-crossref-Linux.tar.xz  work
[nmdemidovich@fedora ~]$ gedit programm4.sh
```

Рис. 4.11: Создание исполняемого файла programm4.sh

```
1 #!/bin/bash
2 format=""
3 directory=""
4 echo "Введите формат файла: "
5 read format
6 echo "Введите директорию: "
7 read directory
8 find "${directory}" -name ".*${format}" -type f | wc -l
9 ls
```

Сохранение файла «/home/nmde... sh ▾ Ширина табуляции: 8 ▾ Стр 9, Стлб 3 ▾ ВСТ

Рис. 4.12: Исполняемый файл programm3.sh

```
nmdemidovich@fedora:~  
[nmdemidovich@fedora ~]$ bash programm4.sh  
Введите формат файла:  
txt  
Введите директорию:  
/home/nmdemidovich  
15  
abc1      io.h      programm1.sh  Видео  
backup    may       programm2.sh  Документы  
bin        newdir.tar programm3.sh  Загрузки  
cat        pandoc-2.19 programm4.sh  Изображения  
conf.txt   pandoc-2.19-linux-amd64 reports       Музыка  
crocum     pandoc-2.19-linux-amd64.tar.gz ski.plases   Общедоступные  
equipment  pandoc-crossref ssh1          'Рабочий стол'  
feathers    pandoc-crossref.1 ssh1.pub     Шаблоны  
file.txt   pandoc-crossref-Linux test.txt  
git.jpg    pandoc-crossref-Linux.tar.xz work  
[nmdemidovich@fedora ~]$
```

Рис. 4.13: Работа исполняемого файла programm4.sh

5 Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

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. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments)-интерфейс переносимой операционной системы для компьютерных сред. Представляет собой набор стандартов, подготовленных институтом инженеров

по электронике и радиотехники (IEEE), который определяет различные аспекты построения операционной системы. POSIX включает такие темы, как программный интерфейс, безопасность, работа с сетями и графический интерфейс. POSIX-совместимые оболочки являются будущим поколением оболочек UNIX и других ОС. Windows NT рекламируется как система, удовлетворяющая POSIX-стандартам. POSIX совместимые оболочки разработаны на базе оболочки Корна; фонд бесплатного программного обеспечения (Free Software Foundation) работает над тем, чтобы и оболочку BASH сделать POSIX-совместимой.

3. Как определяются переменные и массивы в языке программирования bash?

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/andy-ls l myfile > blsls/tmp/andy — ls, ls — l >bls` приведет к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то ее значением является символ пробел. Оболочка bash позволяет создание массивов. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. 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. Каково назначение операторов `let` и `read`?

Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (*term*), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате. Этот формат — `radix#number`, где *radix* (основание системы счисления) - любое число не более 26. Для большинства команд основания систем счисления это - 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток (%). Команда `let` берет два операнда и присваивает их переменной.

5. Какие арифметические операции можно применять в языке программирования `bash`?

Оператор Синтаксис Результат
`!`: `!exp` Если `exp` равно 0, возвращает 1; иначе 0
`!=` `exp1 != exp2`. Если `exp1` не равно `exp2`, возвращает 1; иначе 0
`%` `exp1 % exp2` возвращает остаток от деления `exp1` на `exp2`
`%=` `var=%exp`, присваивает остаток от деления `var` на `exp` переменной `var`
`&` `exp1 & exp2`, возвращает побитовое AND выражений `exp1` и `exp2`
`&&` `exp1 && exp2`. Если и `exp1` и `exp2` не равны нулю, возвращает 1; иначе 0
`&=` `var &= exp`, присваивает `var` побитовое AND переменных `var` и выражения `exp`
`*` `exp1 * exp2`, умножает `exp1` на `exp2`
`=` `var = exp`, умножает `exp` на значение `var` и присваивает результат переменной `var`
`+` `exp1 + exp2`, складывает `exp1` и `exp2`
`+=` `var += exp`, складывает `exp` со значением `var` и результат присваивает `var`
`-` `exp`. Операция отрицания `exp` (называется унарный минус): `- exp1`
`-` `exp1 - exp2`, вычитает `exp2` из `exp1`
`--` `var -= exp`, вычитает `exp` из значения `var` и присваивает результат `var`
`/` `exp1 / exp2`, делит `exp1` на `exp2`
`/=` `var /= exp`, делит `var` на `exp` и

присваивает результат $\text{var} < \text{exp1} < \text{exp2}$. Если exp1 меньше, чем exp2 , возвращает 1, иначе возвращает 0 « $\text{exp1} \ll \text{exp2}$, сдвигает exp1 влево на exp2 бит «= $\text{var} \ll \text{exp}$ Побитовый сдвиг влево значения var на $\text{exp} \leq \text{exp1} \leq \text{exp2}$. Если exp1 меньше, или равно exp2 , возвращает 1; иначе возвращает 0 = $\text{var} = \text{exp}$, присваивает значение exp переменной $\text{va} == \text{exp1} == \text{exp2}$. Если exp1 равно exp2 . Возвращает 1; иначе возвращает 0 > $\text{exp1} > \text{exp2}$ 1 если exp1 больше, чем exp2 ; иначе 0 >= $\text{exp1} >= \text{exp2}$ 1 если exp1 больше, или равно exp2 ; иначе 0 » $\text{exp} \gg \text{exp2}$, сдвигает exp1 вправо на exp2 бит »= $\text{var} \gg \text{exp}$ Побитовый сдвиг вправо значения var на $\text{exp} \wedge \text{exp1} \wedge \text{exp2}$ Исключающее OR выражений. exp1 и $\text{exp2} \wedge \text{var} \wedge \text{exp}$, присваивает var побитовое исключающее OR var и $\text{exp} | \text{exp1} | \text{exp2}$, побитовое OR выражений exp1 и $\text{exp2} |= \text{var} |= \text{exp}$, присваивает var «исключающее OR» переменной var и выражения $\text{exp} || \text{exp1} || \text{exp2}$ 1 если или exp1 или exp2 являются ненулевыми значениями; иначе 0 ~ ~ exp .

6. Что означает операция (())?

Условия оболочки bash.

7. Какие стандартные имена переменных Вам известны?

Имя переменной (идентификатор) — это строка символов, которая отличает эту переменную от других объектов программы (идентифицирует переменную в программе). При задании имен переменным нужно соблюдать следующие правила: § первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются). Можно использовать символ «_»; § в имени нельзя использовать символ «.»; § число символов в имени не должно превышать 255; § имя переменной не должно совпадать с зарезервированными (служебными) словами языка. Var1, PATH, trash, mon, day, PS1, PS2 Другие стандартные переменные: -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. Это символы 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. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

8. Что такое метасимволы?

Такие символы, как ' < > * ? | " & являются метасимволами и имеют для командного процессора специальный смысл.

9. Как экранировать метасимволы?

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, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например, *–echo выведет на экран символ, –echo ab'|'cd* выдаст строку *ab|cd*.

10. Как создавать и запускать командные файлы?

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 ее интерпретацию.

11. Как определяются функции в языке программирования `bash`?

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 иницирует 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. Каким образом можно выяснить, является файл каталогом или обычным файлом?

`ls -lrt` Если есть `d`, то является файл каталогом.

13. Каково назначение команд `set`, `typeset` и `unset`?

Используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, `set -A states Delaware`

Michigan “New Jersey” Далее можно сделать добавление в массив, например, `states[49]=Alaska` . Индексация массивов начинается с нулевого элемента. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды `set`. Наиболее распространенным является сокращение, избавляющееся от слова `let` в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте `typeset -i` для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово `integer` (псевдоним для `typeset -l`) и объявлять переменные целыми. Таким образом, выражения типа `x=y+z` воспринимаются как арифметические. 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 инициализирует 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. В переменные `top` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

14. Как передаются параметры в командные файлы?

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. 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. Назовите специальные переменные языка `bash` и их назначение.

- $\$*$ — 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`-ному элементу массива;
- `${name[*]}` — перечисляет все элементы массива, разделенные пробелом;
- `${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 удаленным самым коротким левым образцом (`pattern`);

- `${#name[*]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.
- `$#` вместо нее будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.

6 Выводы

В результате выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.

Список литературы

Лабораторная работа №10 (Архитектура ОС)