

Отчёт по лабораторной работе 10

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

Сидорова Наталья Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
	Список литературы	22

Список иллюстраций

4.1	скрипт 1	9
4.2	Работа скрипта 1	9
4.3	скрипт 2	10
4.4	Работа скрипта 2	10
4.5	скрипт 3	11
4.6	Работа скрипта 3	11
4.7	скрипт 4	12
4.8	Работа скрипта	12

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 оболочке 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;

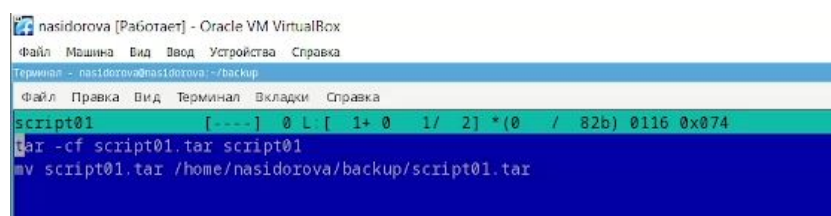
BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

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 Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с

описанными ниже.

4 Выполнение лабораторной работы

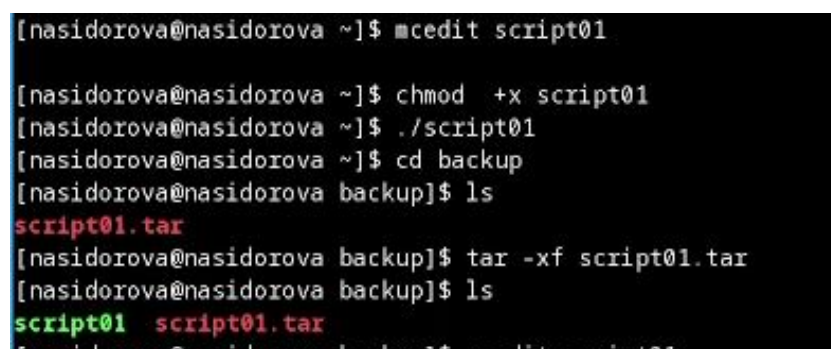
Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. (рис. 4.1).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Терминал - nasidorova@nasidorova: ~/backup
Файл  Правка  Вид  Терминал  Вкладки  Справка
script01  [----]  0 L: [ 1+ 0  1/ 2] *(0  / 82b) 0116 0x074
tar -cf script01.tar script01
mv script01.tar /home/nasidorova/backup/script01.tar
```

Рис. 4.1: скрипт 1

Запуск кода (рис. 4.2).

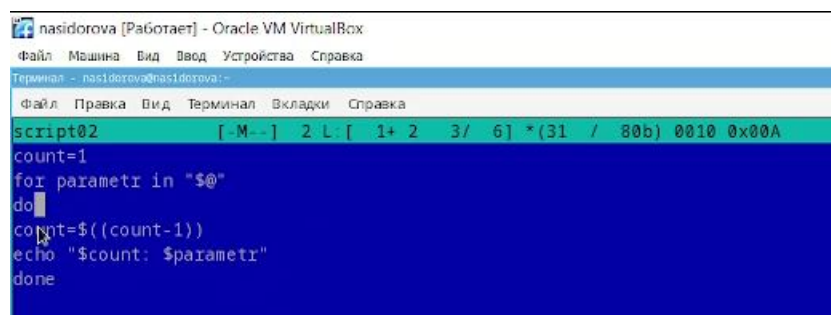


```
[nasidorova@nasidorova ~]$ mcedit script01
[nasidorova@nasidorova ~]$ chmod +x script01
[nasidorova@nasidorova ~]$ ./script01
[nasidorova@nasidorova ~]$ cd backup
[nasidorova@nasidorova backup]$ ls
script01.tar
[nasidorova@nasidorova backup]$ tar -xf script01.tar
[nasidorova@nasidorova backup]$ ls
script01  script01.tar
[nasidorova@nasidorova backup]$ mcedit script01
```

Рис. 4.2: Работа скрипта 1

Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Напри-

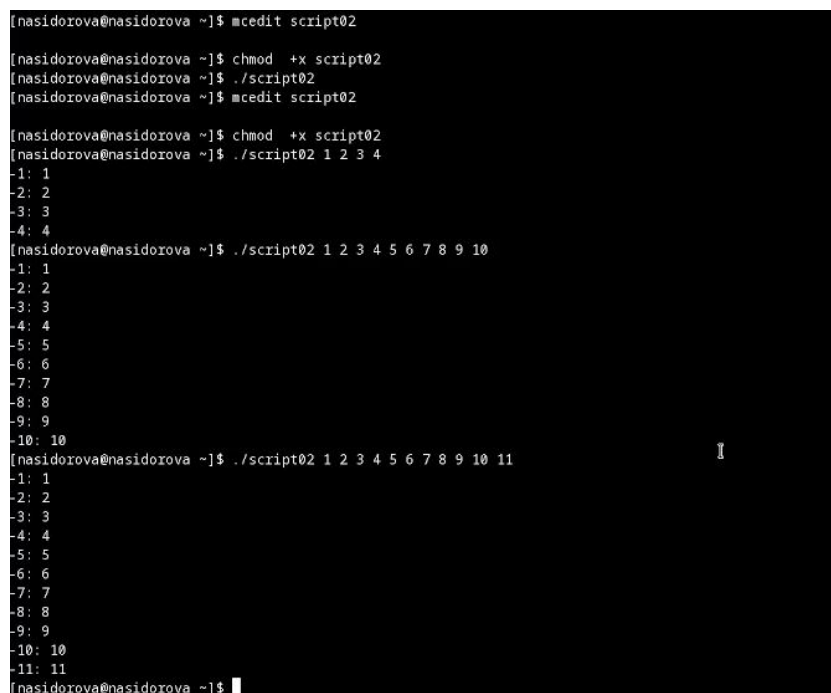
мер, скрипт может последовательно распечатывать значения всех переданных аргументов. (рис. 4.3).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Терминал - nasidorova@nasidorova:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
script02  [-M--]  2 L: [ 1+ 2  3/  6] *(31 / 80b) 0010 0x00A
count=1
for parametr in "$@"
do
count=$((count-1))
echo "$count: $parametr"
done
```

Рис. 4.3: скрипт 2

Запуск кода (рис. 4.4).

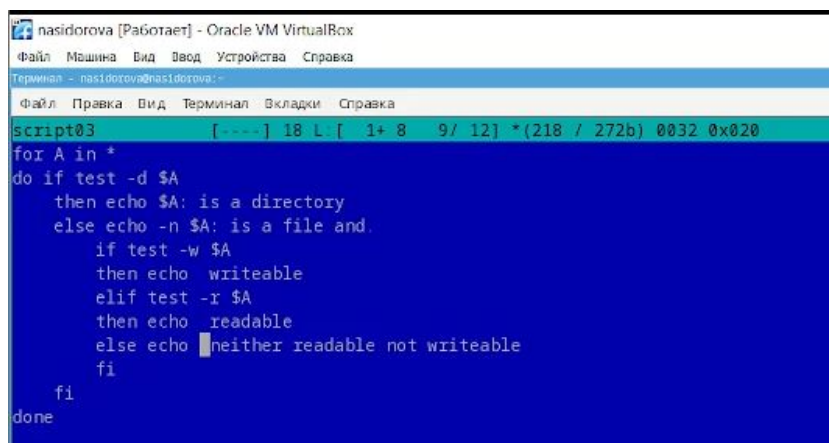


```
[nasidorova@nasidorova ~]$ mcedit script02
[nasidorova@nasidorova ~]$ chmod +x script02
[nasidorova@nasidorova ~]$ ./script02
[nasidorova@nasidorova ~]$ mcedit script02
[nasidorova@nasidorova ~]$ chmod +x script02
[nasidorova@nasidorova ~]$ ./script02 1 2 3 4
-1: 1
-2: 2
-3: 3
-4: 4
[nasidorova@nasidorova ~]$ ./script02 1 2 3 4 5 6 7 8 9 10
-1: 1
-2: 2
-3: 3
-4: 4
-5: 5
-6: 6
-7: 7
-8: 8
-9: 9
-10: 10
[nasidorova@nasidorova ~]$ ./script02 1 2 3 4 5 6 7 8 9 10 11
-1: 1
-2: 2
-3: 3
-4: 4
-5: 5
-6: 6
-7: 7
-8: 8
-9: 9
-10: 10
-11: 11
[nasidorova@nasidorova ~]$
```

Рис. 4.4: Работа скрипта 2

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

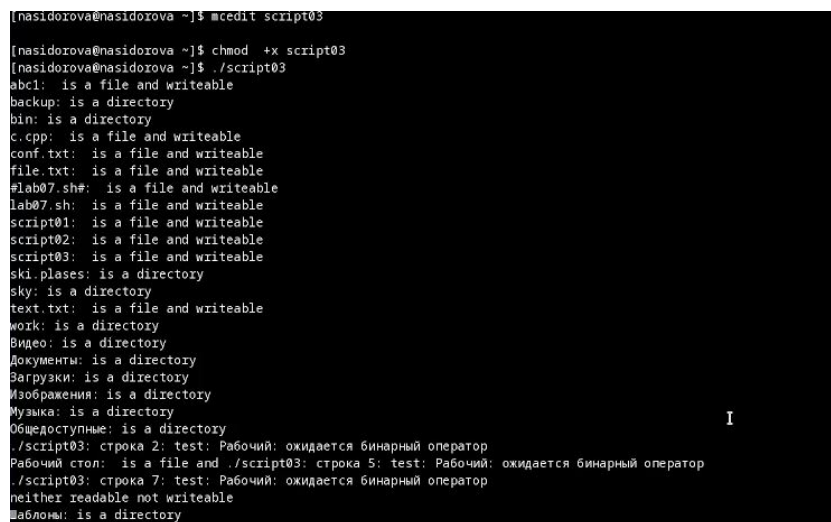
каталога. (рис. 4.5).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Терминал - nasidorova@nasidorova:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
script03  [-----] 18 L [ 1+ 8 9/ 12] *(218 / 272b) 0032 0x020
for A in *
do if test -d $A
then echo $A: is a directory
else echo -n $A: is a file and
if test -w $A
then echo writeable
elif test -r $A
then echo readable
else echo neither readable not writeable
fi
fi
done
```

Рис. 4.5: скрипт 3

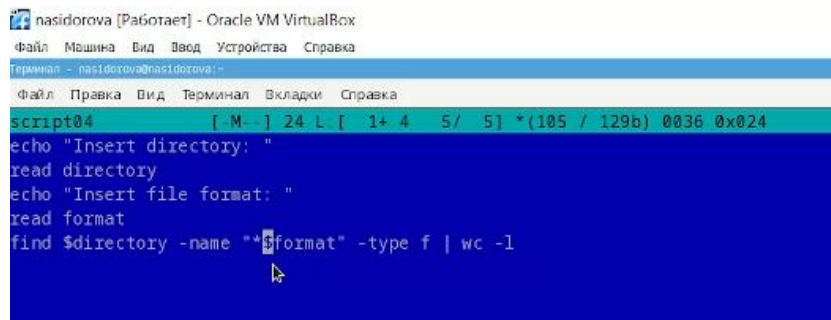
Запуск кода (рис. 4.6).



```
[nasidorova@nasidorova ~]$ mcedit script03
[nasidorova@nasidorova ~]$ chmod +x script03
[nasidorova@nasidorova ~]$ ./script03
abcl: is a file and writeable
backup: is a directory
bin: is a directory
c.cpp: is a file and writeable
conf.txt: is a file and writeable
file.txt: is a file and writeable
#lab07.sh#: is a file and writeable
lab07.sh: is a file and writeable
script01: is a file and writeable
script02: is a file and writeable
script03: is a file and writeable
ski.plases: is a directory
sky: is a directory
text.txt: is a file and writeable
work: is a directory
Видео: is a directory
Документы: is a directory
Загрузки: is a directory
Изображения: is a directory
Музыка: is a directory
Общедоступные: is a directory
./script03: строка 2: test: Рабочий: ожидается бинарный оператор
Рабочий стол: is a file and ./script03: строка 5: test: Рабочий: ожидается бинарный оператор
./script03: строка 7: test: Рабочий: ожидается бинарный оператор
neither readable not writeable
Шаблоны: is a directory
```

Рис. 4.6: Работа скрипта 3

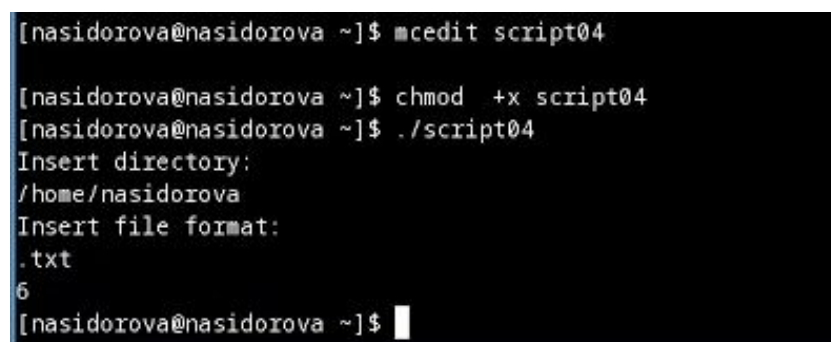
Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки. (рис. 4.7).



```
nasidorova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Терминал - nasidorova@nasidorova:~
script04      [ M-- ] 24 L [ 1+ 4  5/  5] *(105 / 129b) 0036 0x024
echo "Insert directory: "
read directory
echo "Insert file format: "
read format
find $directory -name "$format" -type f | wc -l
```

Рис. 4.7: скрипт 4

Запуск кода (рис. 4.8).



```
[nasidorova@nasidorova ~]$ mcedit script04
[nasidorova@nasidorova ~]$ chmod +x script04
[nasidorova@nasidorova ~]$ ./script04
Insert directory:
/home/nasidorova
Insert file format:
.txt
6
[nasidorova@nasidorova ~]$
```

Рис. 4.8: Работа скрипта

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 надстра-

ивается 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 `{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`? 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

$\&= \text{exp}$ Присваивает var побитовое AND переменных var и выражения exp
 $* \text{exp1} * \text{exp2}$ Умножает exp1 на exp2 $= \text{var} = \text{exp}$ Умножает exp на значение var
 и присваивает результат переменной $\text{var} + \text{exp1} + \text{exp2}$ Складывает exp1 и
 exp2 $+= \text{var} += \text{exp}$ Складывает exp со значением var и результат присваивает
 var $- \text{exp}$ Операция отрицания exp (называется унарный минус) $- \text{exp1} - \text{exp2}$
 Вычитает exp2 из exp1 $-- \text{var} -- \text{exp}$ Вычитает exp из значения var и присваи-
 вает результат $\text{var} / \text{exp} / \text{exp2}$ Делит exp1 на exp2 $/= \text{var} /= \text{exp}$ Делит var на
 exp и присваивает результат $\text{var} < \text{exp1} < \text{exp2}$

Если exp1 меньше, чем exp2 , возвращает 1, иначе возвращает 0 $\ll \text{exp1} \ll \text{exp2}$
 Сдвигает exp1 влево на exp2 бит $\ll= \text{var} \ll= \text{exp}$ Побитовый сдвиг влево значения var
 на exp $\leq \text{exp1} \leq \text{exp2}$ Если exp1 меньше, или равно exp2 , возвра- щает 1; иначе
 возвращает 0 $= \text{var} = \text{exp}$ Присваивает значение exp переменной var $== \text{exp1} == \text{exp2}$
 Если exp1 равно exp2 . Возвращает 1; иначе возвращает 0 $> \text{exp1} > \text{exp2}$ 1 если exp1
 больше, чем exp2 ; иначе 0 $\geq \text{exp1} \geq \text{exp2}$ 1 если exp1 больше, или равно exp2 ;
 иначе 0 $\gg \text{exp} \gg \text{exp2}$ Сдвигает exp1 вправо на exp2 бит $\gg= \text{var} \gg= \text{exp}$ Побитовый
 сдвиг вправо значения var на exp $\wedge \text{exp1} \wedge \text{exp2}$ Исключающее OR

выражений exp1 и exp2 $\wedge= \text{var} \wedge= \text{exp}$ Присваивает var побитовое исключающее
 OR var и exp $| \text{exp1} | \text{exp2}$ Побитовое OR выражений exp1 и exp2 $|= \text{var} |= \text{exp}$
 Присваивает var «исключающее OR» пе- ременной var и выражения exp $|| \text{exp1} ||$
 exp2 1 если или exp1 или exp2 являются нену- левыми значениями; иначе 0 \sim
 $\sim \text{exp}$ Побитовое дополнение до 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 програм-
 ме). 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, раз-

деленных 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 -i`) 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 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. 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 ме-

тасимволом 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 последовательно сти 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. #

Выводы

В ходе выполнения лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.

Список литературы