

Отчёт по лабораторной работе №12

Операционные системы

Андреева Софья Владимировна

Содержание

1	Цель работы	4
2	Выполнение работы	5
3	Контрольные вопросы	9
4	Выводы	18

Список иллюстраций

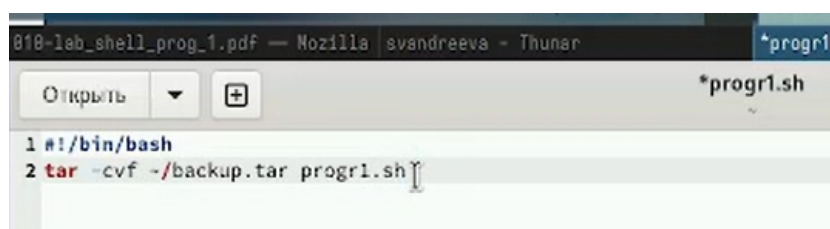
2.1	Скрипт	5
2.2	Скрипт	5
2.3	Значения всех переданных аргументов	6
2.4	Скрипт	7
2.5	Командный файл — аналог команды ls	7
2.6	Скрипт	8
2.7	Количество файлов в указанной директории	8

1 Цель работы

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

2 Выполнение работы

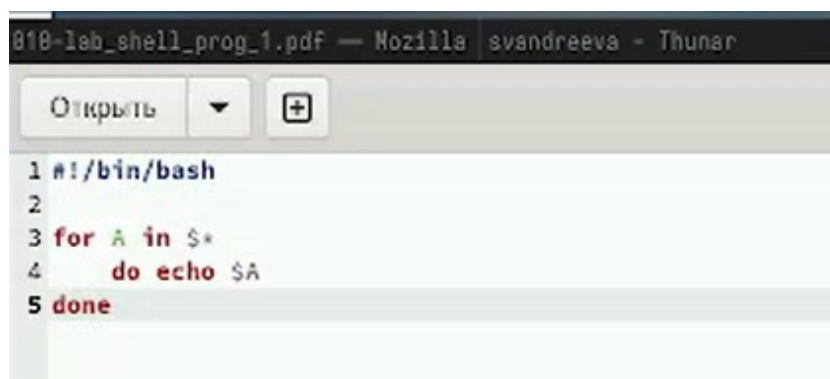
Написала скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. (рис. fig. 2.1).

A screenshot of a terminal window titled "818-lab_shell_prog_1.pdf — Mozilla | svandreeva - Thunar". The terminal shows a shell script with two lines: "1 #!/bin/bash" and "2 tar -cvf ~/backup.tar progr1.sh". The cursor is at the end of the second line.

```
1 #!/bin/bash
2 tar -cvf ~/backup.tar progr1.sh
```

Рис. 2.1: Скрипт

Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт может последовательно распечатывать значения всех переданных аргументов (рис. fig. 2.2),(рис. fig. 2.3).

A screenshot of a terminal window titled "818-lab_shell_prog_1.pdf — Mozilla | svandreeva - Thunar". The terminal shows a shell script with five lines: "1 #!/bin/bash", "2", "3 for A in \$*", "4 do echo \$A", and "5 done".

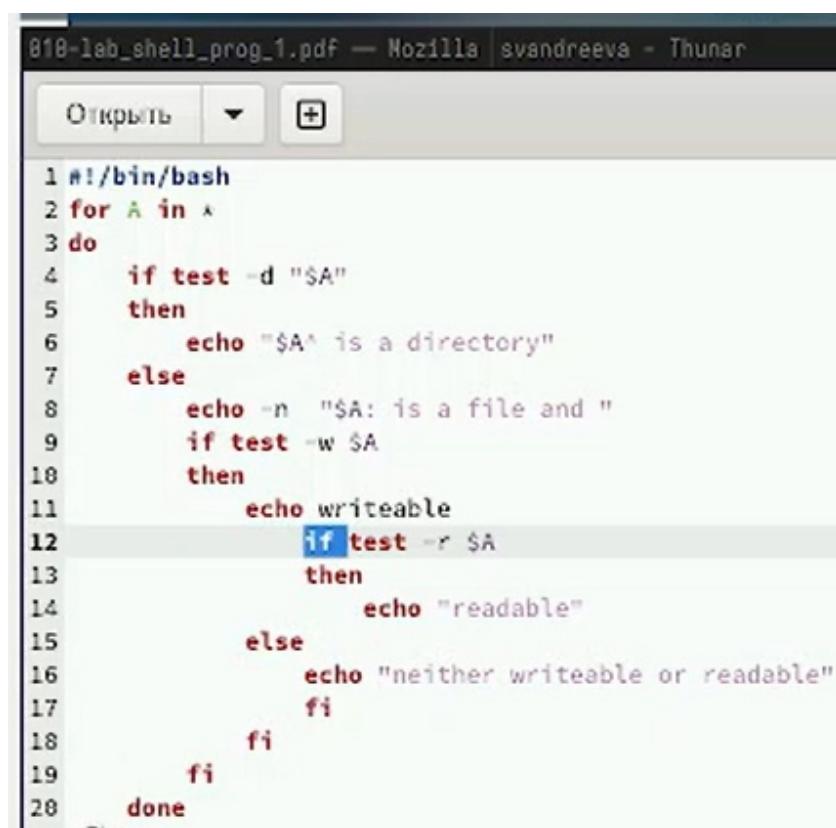
```
1 #!/bin/bash
2
3 for A in $*
4     do echo $A
5 done
```

Рис. 2.2: Скрипт

```
[svandreeva@fedora ~]$ bash progr2.sh 33 fg 4 000 8989 j8 i 44 e3 k9
33
fg
4
000
8989
j8
i
44
e3
k9
[svandreeva@fedora ~]$ bash progr2.sh 33 fg 4 000 8989 j8 i 44 e3 k9 hh
33
fg
4
000
8989
j8
i
44
e3
k9
hh
```

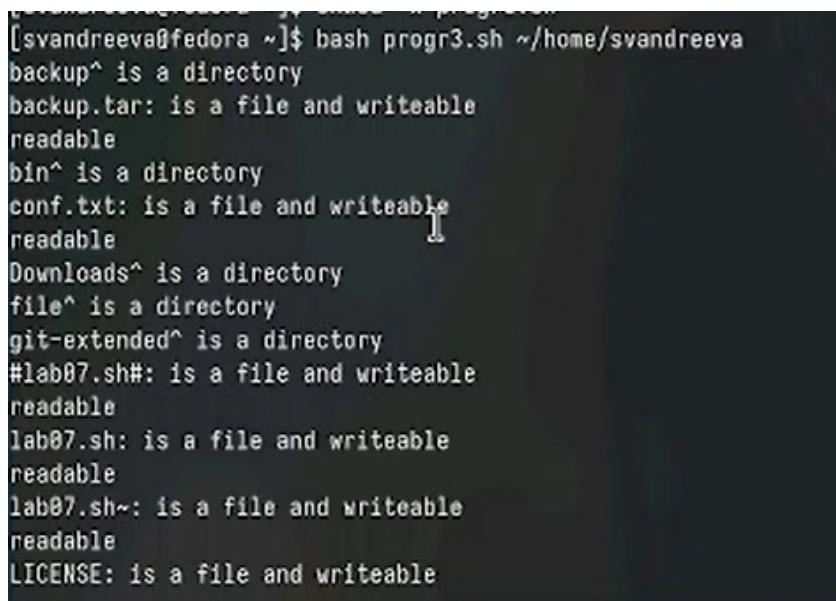
Рис. 2.3: Значения всех переданных аргументов

Написала командный файл — аналог команды `ls`. Он выдает информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога(рис. fig. 2.4), (рис. fig. 2.5).

A screenshot of a text editor window titled '010-lab_shell_prog_1.pdf — Mozilla | svandreeva - Thunar'. The editor contains a shell script with 20 lines of code. The script starts with a shebang line, followed by a 'for' loop that iterates over all files in the current directory. Inside the loop, it uses 'if test' to check if a file is a directory. If it is, it prints a message. If not, it checks if the file is writable and readable, printing appropriate messages. The script ends with a 'done' statement.

```
1 #!/bin/bash
2 for A in *
3 do
4     if test -d "$A"
5     then
6         echo "$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 writeable or readable"
17        fi
18    fi
19 fi
20 done
```

Рис. 2.4: Скрипт

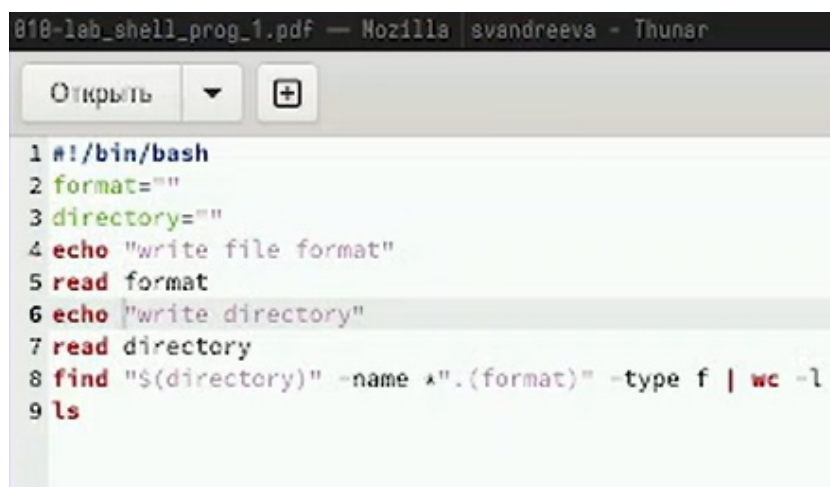
A screenshot of a terminal window showing the execution of the script from Figure 2.4. The prompt is '[svandreeva@fedora ~]\$'. The command executed is 'bash progr3.sh ~/home/svandreeva'. The output lists various files and directories, indicating whether they are directories, files, writeable, and readable. A cursor is visible on the line 'conf.txt: is a file and writeable readable'.

```
[svandreeva@fedora ~]$ bash progr3.sh ~/home/svandreeva
backup^ is a directory
backup.tar: is a file and writeable
readable
bin^ is a directory
conf.txt: is a file and writeable
readable
Downloads^ is a directory
file^ is a directory
git-extended^ is a directory
#lab07.sh#: is a file and writeable
readable
lab07.sh: is a file and writeable
readable
lab07.sh~: is a file and writeable
readable
LICENSE: is a file and writeable
```

Рис. 2.5: Командный файл — аналог команды ls

Написала командный файл, который получает в качестве аргумента команд-

ной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (рис. fig. 2.6) (рис. fig. 2.7).



```
010-lab_shell_prog_1.pdf — Mozilla | svandreeva - Thunar
Открыть ▼ +
1 #!/bin/bash
2 format=""
3 directory=""
4 echo "write file format"
5 read format
6 echo "write directory"
7 read directory
8 find "${directory}" -name "*.${format}" -type f | wc -l
9 ls
```

Рис. 2.6: Скрипт



```
[svandreeva@fedora ~]$ bash progr4.sh
write file format
txt
write directory
/home/svandreeva
33
[svandreeva@fedora ~]$
```

Рис. 2.7: Количество файлов в указанной директории

3 Контрольные вопросы

- 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:
 1. Оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций;
 2. C-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд;
 3. оболочка Корна - напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
 4. BASH - сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).
- 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.

- 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 мета символ.
- Использование команд `b=/tmp/andy-ls -l myfile >pblls/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.

- 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.
- 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 `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 - exp2` Вычитает `exp2` из `exp1`
`-- var -- exp` Вычитает `exp` из значения `var` и присваивает результат `var`
`/ exp / exp2` Делит `exp1` на `exp2`
`/= var /= exp` Делит `var` на `exp` и присваивает результат `var`
`< exp1 < exp2` Если `exp1` меньше, чем `exp2`, возвращает 1, иначе возвращает 0
`<< exp1 << exp2` Сдвигает `exp1` влево на `exp2` бит
`<= var <= exp` Побитовый сдвиг влево значения `var` на `exp`

\leq expr2 Если expr1 меньше, или равно expr2, возвращает 1; иначе возвращает 0
 $=$ var = expr Присваивает значение expr переменной var
 $==$ expr1 == expr2 Если expr1 равно expr2. Возвращает 1; иначе возвращает 0
 $>$ expr1 > expr2 1 если expr1 больше, чем expr2; иначе 0
 \geq expr1 \geq expr2 1 если expr1 больше, или равно expr2; иначе 0
 \gg expr \gg expr2 Сдвигает expr1 вправо на expr2 бит
 $\gg=$ var $\gg=$ expr Побитовый сдвиг вправо значения var на expr
 \wedge expr1 \wedge expr2 Исключающее OR выражений expr1 и expr2
 $\wedge=$ var $\wedge=$ expr Присваивает var побитовое исключающее OR var и expr
 $|$ expr1 $|$ expr2 Побитовое OR выражений expr1 и expr2
 $|=$ var $|=$ expr Присваивает var «исключающее OR» переменной var и выражения expr
 $||$ expr1 $||$ expr2 1 если или expr1 или expr2 являются ненулевыми значениями; иначе 0
 \sim expr Побитовое дополнение до expr.

- Что означает операция (())? Условия оболочки bash.
- 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.

Что такое метасимволы? Такие символы, как ' < > * ? | " & являются мета- символами и имеют для командного процессора специальный смысл.

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.

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.

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. Каким образом можно выяснить, является файл каталогом или обычным файлом? `ls -lrf` Если есть `d`, то является файл каталогом

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 -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 спи-

сок 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 `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

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.

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.

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; 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 выполне

4 Выводы

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