

Отчёт по лабораторной работе №11

Операционные системы

Екатерина Павловна Канева

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	15
6	Контрольные вопросы	16

Список иллюстраций

4.1	Программа 1.	9
4.2	Файл input.	9
4.3	Запуск программы.	9
4.4	Файл output.	10
4.5	Программа 2 (.c).	11
4.6	Программа 2 (.sh).	11
4.7	Запуск программы.	12
4.8	Программа 3.	13
4.9	Запуск программы.	13
4.10	Программа 4.	13
4.11	Запуск программы.	14
4.12	Файл output.	14

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-р`шаблон — указать шаблон для поиска;
- `-С` — различать большие и малые буквы;
- `-n` — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом `-р`. 2. Написать на языке Си программу, которая вводит число и определяет, является ли оно большим нуля, меньшим нуля или равным нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. 3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например `1.tmp`, `2.tmp`, `3.tmp`, `4.tmp` и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). 4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`).

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.

4 Выполнение лабораторной работы

1. Создала программу, требуемую заданием 1 (рис. 4.1) и проверила её работу (рис. 4.2, 4.3 и 4.4):

```
#!/bin/bash

while getopts i:o:p:cn optletter
do
case $optletter in
    i) iflag=1; ival=$OPTARG;;
    o) oflag=1; oval=$OPTARG;;
    p) pflag=1; pval=$OPTARG;;
    c) cflag=1;;
    n) nflag=1;;
    *) echo $optletter is illegal option
    esac
done

if ! test cflag
then
    cf=-i
fi

if test nflag
```


then

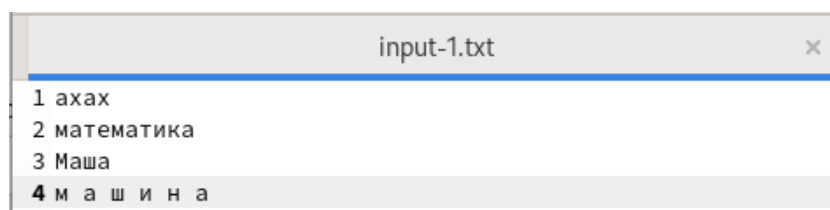
`nf=-n`

fi

`grep $cf $nf $pval $ival >> $oval`

```
1 #!/bin/bash
2
3 while getopts i:o:p:cn optletter
4 do
5     case $optletter in
6         i) iflag=1; ival=$OPTARG;;
7         o) oflag=1; oval=$OPTARG;;
8         p) pflag=1; pval=$OPTARG;;
9         c) cflag=1;;
10        n) nflag=1;;
11        *) echo $optletter is illegal option
12           esac
13    done
14
15    if ! test cflag
16    then
17        cf=-i
18    fi
19
20    if test nflag
21    then
22        nf=-n
23    fi
24
25    grep $cf $nf $pval $ival >> $oval
```

Рис. 4.1: Программа 1.



input-1.txt

```
1 ахах
2 математика
3 Маша
4 м а ш и н а
```

Рис. 4.2: Файл input.

```
[epkaneva@epkaneva ~]$ touch 1.sh
[epkaneva@epkaneva ~]$ gedit 1.sh
[epkaneva@epkaneva ~]$ chmod +x 1.sh
[epkaneva@epkaneva ~]$ touch input-1.txt
[epkaneva@epkaneva ~]$ gedit input-1.txt &
[1] 3212
[epkaneva@epkaneva ~]$ bash 1.sh -p ма -i input-1.txt -o output.txt -c -n
```

Рис. 4.3: Запуск программы.

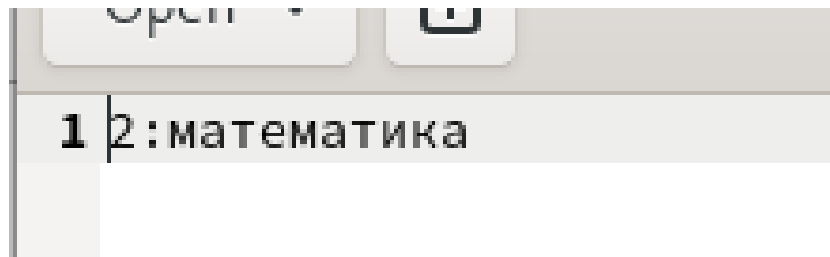


Рис. 4.4: Файл output.

2. Создала программы, требуемые заданием 2 (рис. 4.5 и 4.6) и проверила их работу (рис. 4.7):

```
#include <stdlib.h>
#include <stdio.h>

int main() {
    int n;

    printf("Input a whole number: ");
    scanf("%d", &n);

    if (n == 0) {
        exit(0);
    } else if (n > 0) {
        exit(1);
    } else {
        exit(2);
    }
}

#!/bin/bash

gcc -o 2 2.c
```

./2

```
case $? in
    0) echo "The number equals 0.>";
    1) echo "The number is above 0.>";
    2) echo "The number is below 0.>";
esac
```

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main() {
5     int n;
6
7     printf("Input a whole number: ");
8     scanf("%d", &n);
9
10    if (n == 0) {
11        exit(0);
12    } else if (n > 0) {
13        exit(1);
14    } else {
15        exit(2);
16    }
17 }
```

Рис. 4.5: Программа 2 (.c).

```
2.sh
1 #!/bin/bash
2
3 gcc -o 2 2.c
4 ./2
5
6 case $? in
7     0) echo "The number equals 0.>";
8     1) echo "The number is above 0.>";
9     2) echo "The number is below 0.>";
10 esac
```

Рис. 4.6: Программа 2 (.sh).

```

[epkaneva@epkaneva ~]$ touch 2.sh
[2]+  Done                  gedit 1.sh
[epkaneva@epkaneva ~]$ touch 2.c
[epkaneva@epkaneva ~]$ gedit 2.sh 2.c &
[1] 3393
[epkaneva@epkaneva ~]$ chmod +x 2.sh
[epkaneva@epkaneva ~]$ bash 2.sh
Input a whole number: 4
The number is above 0.
[epkaneva@epkaneva ~]$ bash 2.sh
Input a whole number: 0
The number equals 0.
[epkaneva@epkaneva ~]$ bash 2.sh
Input a whole number: -123
[epkaneva@epkaneva ~]$ bash 2.sh
Input a whole number: -123
The number is below 0.

```

Рис. 4.7: Запуск программы.

3. Создала программу, требуемую заданием 3 (рис. 4.8) и проверила её работу (рис. 4.9):

```
#!/bin/bash
```

```

for ((i=1; i<=$*; i++))
do
    if test -f "$i".tmp
    then rm "$i".tmp
    else touch "$i".tmp
    fi
done

```

```

1 #!/bin/bash
2
3 for ((i=1; i<=$*; i++))
4 do
5     if test -f "$i".tmp
6     then rm "$i".tmp
7     else touch "$i".tmp
8     fi
9 done

```

Рис. 4.8: Программа 3.

```

[epkaneva@epkaneva ~]$ bash 3.sh 3
[epkaneva@epkaneva ~]$ ls
1.sh  2  2.sh  3.sh  4.sh  bin  Documents  input-1.txt  output.txt  Public  Videos
1.tmp  2.c  2.tmp  3.tmp  backup Desktop  Downloads  Music  Pictures  Templates work
[epkaneva@epkaneva ~]$ bash 3.sh 3
[epkaneva@epkaneva ~]$ ls
1.sh  2.c  3.sh  backup Desktop  Downloads  Music  Pictures  Templates work
2  2.sh  4.sh  bin  Documents  input-1.txt  output.txt  Public  Videos

```

Рис. 4.9: Запуск программы.

4. Создала программу, требуемую заданием 4 (рис. 4.10) и проверила её работу (рис. 4.11 и 4.12):

```
#!/bin/bash
```

```
find $* -mtime -7 mtime +0 -type f > FILES.txt
```

```
tar -cf archive.tar -T FILES.txt
```

```

1 #!/bin/bash
2
3 find $* -mtime -7 mtime +0 -type f > FILES.txt
4 tar -cf archive.tar -T FILES.txt

```

Рис. 4.10: Программа 4.

```

[epkaneva@epkaneva ~]$ bash 4.sh /home/epkaneva
find: paths must precede expression: `mtime'
[epkaneva@epkaneva ~]$ gedit FILES.txt &
[2] 3910
[epkaneva@epkaneva ~]$ ls
1.sh  2.c  3.sh  archive.tar  bin  Documents  FILES.txt  Music  Pictures  Templates  work
2     2.sh  4.sh  backup      Desktop  Downloads  input-1.txt  output.txt  Public  Videos

```

Рис. 4.11: Запуск программы.

```

1 | /home/epkaneva/1.sh
2 /home/epkaneva/2.sh
3 /home/epkaneva/2.c
4 /home/epkaneva/3.sh
5 /home/epkaneva/4.sh
6 /home/epkaneva/input-1.txt
7 /home/epkaneva/output.txt

```

Рис. 4.12: Файл output.

5 Выводы

Изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

6 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.

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
`!` `expr` Если `expr` равно 0, возвращает 1; иначе 0
`!=` `expr1 != expr2` Если `expr1` не равно `expr2`, возвращает 1; иначе 0
`%` `expr1 % expr2` Возвращает остаток от деления `expr1` на `expr2`
`%=` `var=%expr` Присваивает остаток от деления `var` на `expr` переменной `var`
`&` `expr1 & expr2` Возвращает побитовое AND выражений `expr1` и `expr2`
`&&` `expr1 && expr2` Если и `expr1` и `expr2` не равны нулю, возвращает 1; иначе 0
`&=` `var &= expr` Присваивает `var` побитовое AND переменных `var` и выражения `expr`
`*` `expr1 * expr2` Умножает `expr1` на `expr2`
`=` `var = expr` Умножает `expr` на значение `var` и присваивает результат переменной `var`
`+` `expr1 + expr2` Складывает `expr1` и `expr2`
`+=` `var += expr` Складывает `expr` со значением `var` и результат присваивает `var`
`-` `expr1 - expr2` Операция отрицания `expr` (называется унарный минус)
`--` `var -- expr` Вычитает `expr2` из `expr1`
`/` `var / expr` Делит `expr1` на `expr2`
`/=` `var /= expr` Делит `var` на `expr` и присваивает результат `var`
`<` `expr1 < expr2`

Если `expr1` меньше, чем `expr2`, возвращает 1, иначе возвращает 0
`<=` `var <= expr` Сдвигает `expr1` влево на `expr2` бит
`<=` `var <= expr` Побитовый сдвиг влево значения `var` на `expr`
`<=` `expr1 <= 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
`>=` `expr1 >= expr2` 1 если `expr1` больше, или равно `expr2`; иначе 0
`>=` `var >= expr` Сдвигает `expr1` вправо на `expr2` бит
`>=` `var >= expr` Побитовый сдвиг вправо значения `var` на `expr`
`^` `expr1 ^ expr2` Исключающее OR выражений

`exp1` и `exp2` `^= var ^= exp` Присваивает `var` побитовое исключающее OR `var` и `exp`
`| exp1 | exp2` Побитовое OR выражений `exp1` и `exp2` `|= var |= exp` Присваивает `var`
«исключающее OR» переменной `var` и выражения `exp` `|| exp1 || exp2` 1 если или
`exp1` или `exp2` являются ненулевыми значениями; иначе 0 `~ ~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 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 `mon` 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.