

Отчёт по лабораторной работе 10

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

Ангелина Дмитриевна Чванова

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	15
6	Контрольные вопросы	16

Список иллюстраций

4.1	Создание файла и изучение справки о архиваторах, открытие емас в фоновом режиме	9
4.2	Скрипт программы	9
4.3	Команда <code>chmod +x</code> , которая дает право на исполнение	10
4.4	Запуск программы и проверка создания архива.	10
4.5	Создание файла, открытие емас в фоновом режиме, добавление прав на исполнение и запуск программы	11
4.6	Скрипт программы	11
4.7	Создание файла, открытие емас в фоновом режиме	12
4.8	Скрипт программы	12
4.9	Запуск программы	13
4.10	Создание файла, открытие емас в фоновом режиме, добавление прав на исполнение	13
4.11	Скрипт программы	14
4.12	Запуск программы	14

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

Здесь описываются теоретические аспекты, связанные с выполнением работы. 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.

Оболочка `bash` поддерживает встроенные арифметические функции. 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 типа `radix#number`, где `radix` (основание системы счисления) — любое число не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

Стандартные переменные:

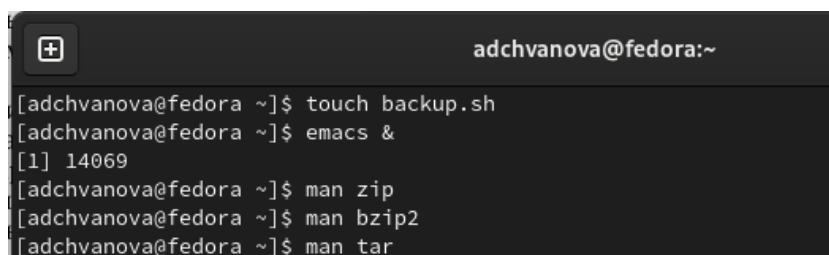
- `HOME` — имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.

– IFS — последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line). – MAIL — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта). – TERM — тип используемого терминала. – LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

4 Выполнение лабораторной работы

1. 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.

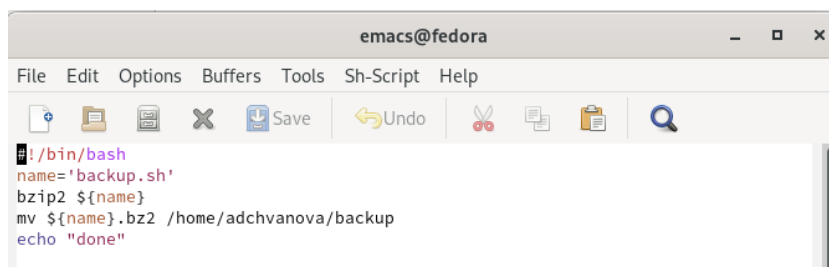
Создаем файл и изучаем архиваторы командой man, а также открываем emacs в фоновом режиме. (рис. 4.1)



```
adchvanova@fedora:~  
[adchvanova@fedora ~]$ touch backup.sh  
[adchvanova@fedora ~]$ emacs &  
[1] 14069  
[adchvanova@fedora ~]$ man zip  
[adchvanova@fedora ~]$ man bzip2  
[adchvanova@fedora ~]$ man tar
```

Рис. 4.1: Создание файла и изучение справки о архиваторах, открытие emacs в фоновом режиме

Пишем программу.(рис. 4.2)



```
emacs@fedora  
File Edit Options Buffers Tools Sh-Script Help  
[Icons]  
#!/bin/bash  
name='backup.sh'  
bzip2 ${name}  
mv ${name}.bz2 /home/adchvanova/backup  
echo "done"
```

Рис. 4.2: Скрипт программы

Делаем файл исполняемым. (рис. 4.3)

```
[adchvanova@fedora ~]$ chmod +x backup.sh
```

Рис. 4.3: Команда `chmod +x`, которая дает право на исполнение

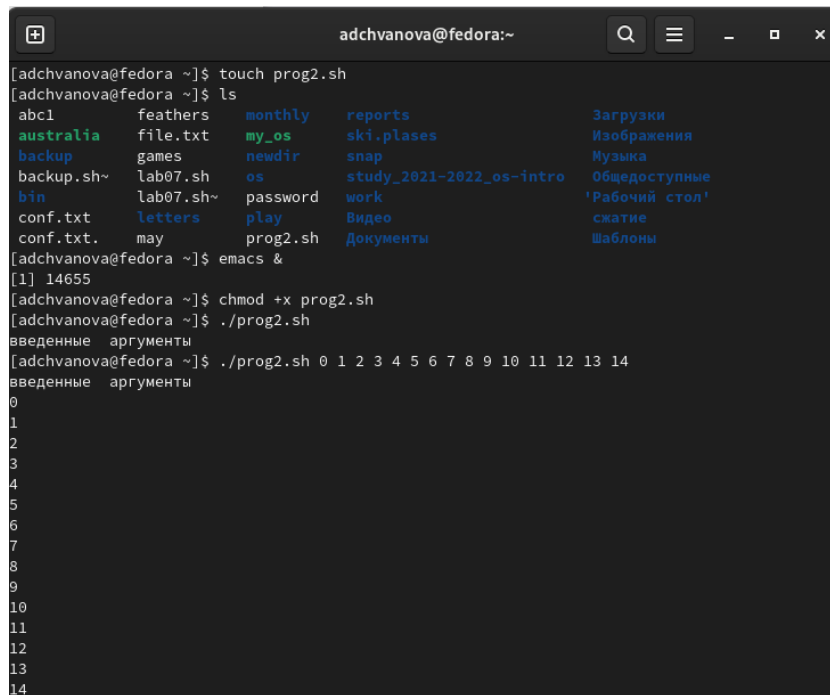
Запускаем программу и проверяем создание архива. (рис. 4.4)

```
[adchvanova@fedora ~]$ cd backup/  
[adchvanova@fedora backup]$ ls  
backup.sh.bz2
```

Рис. 4.4: Запуск программы и проверка создания архива.

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.

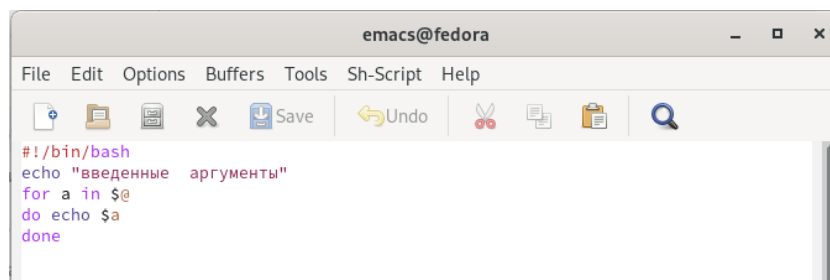
Создаем файл, а также открываем `etacs` в фоновом режиме. После написания программы делаем файл исполняемым и запускаем. (рис. 4.5)



```
[adchvanova@fedora ~]$ touch prog2.sh
[adchvanova@fedora ~]$ ls
abc1      feathers  monthly  reports  Загрузки
australia file.txt  my_os    ski.places  Изображения
backup    games    newdir   snap       Музыка
backup.sh lab07.sh os        study_2021-2022_os-intro  Общедоступные
bin       lab07.sh password work       'Рабочий стол'
conf.txt  letters  play     Видео      сжатие
conf.txt. may      prog2.sh Документы  Шаблоны
[adchvanova@fedora ~]$ emacs &
[1] 14655
[adchvanova@fedora ~]$ chmod +x prog2.sh
[adchvanova@fedora ~]$ ./prog2.sh
введенные аргументы
[adchvanova@fedora ~]$ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
введенные аргументы
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Рис. 4.5: Создание файла, открытие emacs в фоновом режиме, добавление прав на исполнение и запуск программы

Пишем программу.(рис. 4.6)



```
#!/bin/bash
echo "введенные аргументы"
for a in $@
do echo $a
done
```

Рис. 4.6: Скрипт программы

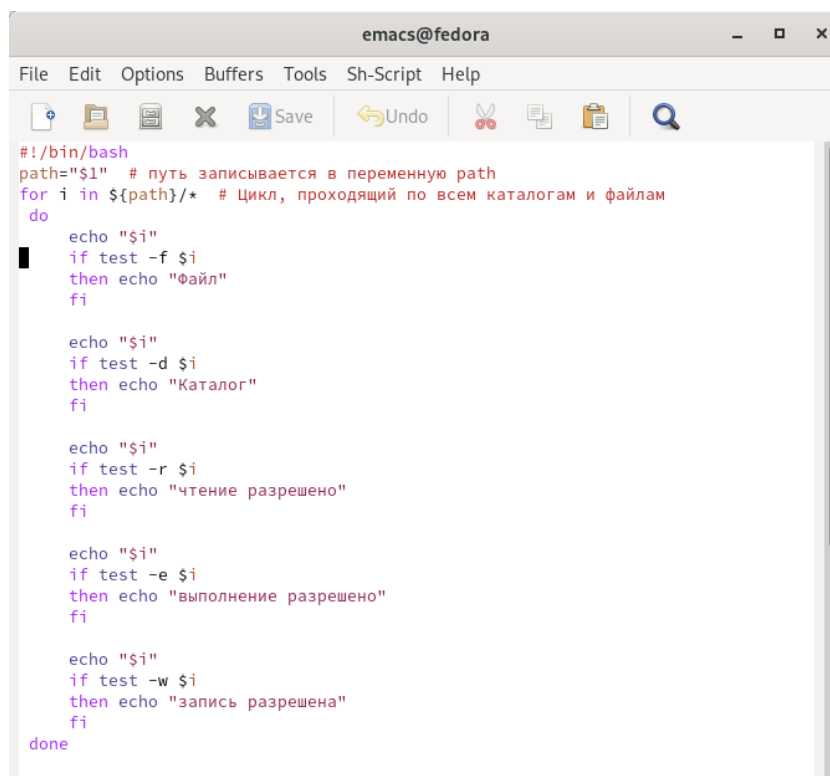
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

Создаем файл , а также открываем emacs в фоновом режиме. (рис. 4.7)

```
[adchvanova@fedora ~]$ touch prog3.sh
[1]+  Завершён      emacs
[adchvanova@fedora ~]$ emacs &
```

Рис. 4.7: Создание файла, открытие emacs в фоновом режиме

Пишем программу.(рис. 4.8)



The screenshot shows the Emacs editor window titled 'emacs@fedora'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The toolbar contains icons for new file, open, save, close, save all, undo, redo, copy, paste, and search. The main text area contains a shell script with the following content:

```
#!/bin/bash
path="$1" # путь записывается в переменную path
for i in ${path}/* # Цикл, проходящий по всем каталогам и файлам
do
    echo "$i"
    if test -f $i
    then echo "Файл"
    fi

    echo "$i"
    if test -d $i
    then echo "Каталог"
    fi

    echo "$i"
    if test -r $i
    then echo "чтение разрешено"
    fi

    echo "$i"
    if test -e $i
    then echo "выполнение разрешено"
    fi

    echo "$i"
    if test -w $i
    then echo "запись разрешена"
    fi
done
```

Рис. 4.8: Скрипт программы

Делаем файл исполняемым. Запускаем программу (рис. 4.9)

```

[adchvanova@fedora ~]$ ./prog3.sh ~
/home/adchvanova/abc1
Файл
/home/adchvanova/abc1
/home/adchvanova/abc1
чтение разрешено
/home/adchvanova/abc1
выполнение разрешено
/home/adchvanova/abc1
запись разрешена
/home/adchvanova/australia
Файл
/home/adchvanova/australia
/home/adchvanova/australia
чтение разрешено
/home/adchvanova/australia
выполнение разрешено
/home/adchvanova/australia
запись разрешена
/home/adchvanova/backup
/home/adchvanova/backup
Каталог
/home/adchvanova/backup
чтение разрешено

```

Рис. 4.9: Запуск программы

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.

Создаем файл , а также открываем emacs в фоновом режиме. После написания программы добавляем права на исполнение (рис. 4.10)

```

[adchvanova@fedora ~]$ touch prog4.sh
[1]+  Завершён          emacs
[adchvanova@fedora ~]$ emacs $
^C[adchvanova@fedora ~]$ chmod +x prog4.sh

```

Рис. 4.10: Создание файла, открытие emacs в фоновом режиме, добавление прав на исполнение

Пишем программу.(рис. 4.11)

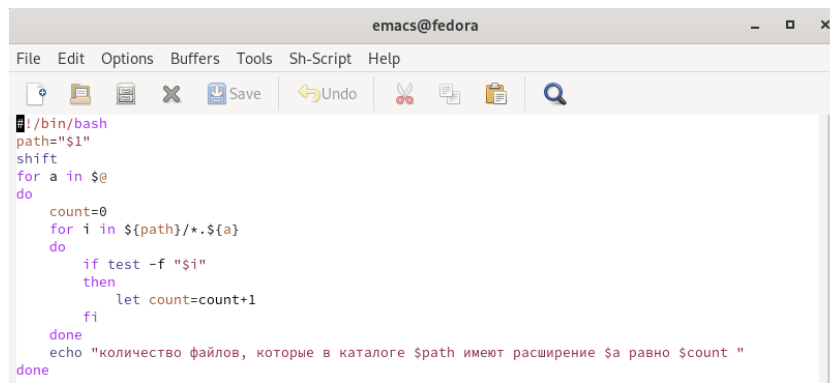


Рис. 4.11: Скрипт программы

Запускаем программу (рис. 4.12)

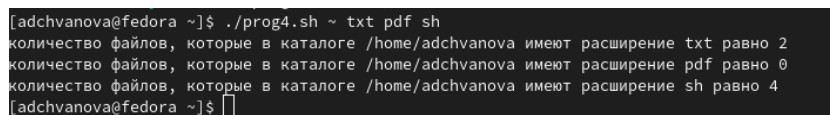


Рис. 4.12: Запуск программы

5 Выводы

Мы изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы. (скрипт, который при запуске делает резервную копию самого себя; скрипт, обрабатывающий любое произвольное число аргументов командной строки; командный файл — аналог команды `ls`; 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)

6 Контрольные вопросы

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

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;
- С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — 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) — 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

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`. Оболочка bash позволяет работать с массивами. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «`set -A states Delaware Michigan "New Jersey"`». Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.

4. 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. Команда `let` берет два операнда и присваивает их переменной. Команда `read` позволяет читать значения переменных со стандартного ввода: «`echo "Please enter Month and Day of Birth ?"`» «`read mon day trash`». В переменные `mon` и `day` будут считаны соответствующие значения, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.

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

В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление

(/) и целочисленный остаток от деления (%).

6.Что означает операция (())?

В (()) записывают условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7.Какие стандартные имена переменных Вам известны?

– HOME — имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
– IFS — последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).
– MAIL — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение `You have mail` (у Вас есть почта).
– TERM — тип используемого терминала.
– LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8.Что такое метасимволы?

`' < > * ? | " &`, 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 нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$`, `'`, `,`, `"`. Например, `-echo*` выведет на экран символ `,`, `-echoab'|` выведет на экран 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.

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, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`

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

«`test -f [путь до файла]`» (для проверки, является ли обычным файлом)

«`test -d [путь до файла]`» (для проверки, является ли каталогом)

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

«`set`» можно использовать для вывода списка переменных окружения. В системах `Ubuntu` и `Debian` команда «`set`» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «`set | more`». Команда «`typeset`» предназначена для наложения ограничений на переменные. Команду «`unset`» следует использовать для удаления переменной из окружения командной оболочки.

14. Как передаются параметры в командные файлы?

При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ `$`

является метасимволом командного процессора.

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

Специальные переменные: 1. `$*` – отображается вся командная строка или параметры оболочки; 2. `$?` – код завершения последней выполненной команды; 3. `$ (2 Таких знака)` – уникальный идентификатор процесса, в рамках которого выполняется командный процессор; 4. `$!` – номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; 5. `$` значение флагов командного процессора; 6. `${#}` – *возвращает целое число – количество слов, которые были результатом \$*; 7. `${#name}` – возвращает целое значение длины строки в переменной `name`; 8. `${name[n]}` – обращение к `n`-му элементу массива; 9. `${name[*]}` – перечисляет все элементы массива, разделённые пробелом; 10. `${name[@]}` – то же самое, но позволяет учитывать символы пробелы в самих переменных; 11. `${name:-value}` – если значение переменной `name` не определено, то оно будет заменено на указанное `value`; 12. `${name:value}` – проверяется факт существования переменной; 13. `${name=value}` – если `name` не определено, то ему присваивается значение `value`; 14. `${name?value}` – останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке; 15. `${name+value}` – это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`; 16. `${name#pattern}` – представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`); 17. `${#name[*]}` и `${#name[@]}` – эти выражения возвращают количество элементов в массиве `name`.