

# **Лабораторная работа №10**

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

Крутова Екатерина Дмитриевна

# Содержание

<b>1</b>	<b>Цель работы:</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Задание 1 (рис. 4.1-4.4) . . . . .	8
4.2	Задание 2 (рис. 4.5-4.7) . . . . .	9
4.3	Задание 3 (рис. 4.8-4.9) . . . . .	11
4.4	Задание 4 (рис. 4.10-4.11) . . . . .	12
<b>5</b>	<b>Выводы</b>	<b>13</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>14</b>

## Список иллюстраций

4.1	Просмотр справки . . . . .	8
4.2	Создание файла и директории . . . . .	8
4.3	Код программы . . . . .	9
4.4	Преобразование и исполнение . . . . .	9
4.5	Создание файла . . . . .	9
4.6	Код программы . . . . .	10
4.7	Преобразование и исполнение . . . . .	10
4.8	Код программы . . . . .	11
4.9	Преобразование и исполнение . . . . .	11
4.10	Код программы . . . . .	12
4.11	Преобразование и исполнение . . . . .	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;
- 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).

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

### 4.1 Задание 1 (рис. 4.1-4.4)

```
[edkrutova@fedora ~]$ man zip
[edkrutova@fedora ~]$ man bzip2
[edkrutova@fedora ~]$ man tar
[edkrutova@fedora ~]$
```

Рис. 4.1: Просмотр справки

```
[edkrutova@fedora ~]$ mkdir backup
[edkrutova@fedora ~]$ ls
abc1      feathers  'main (14).cpp'  name_h     ski.places  Документы  Общедоступные
australia file.txt  '#Messages+#'    name_h.txt text.txt    Загрузки   'Рабочий стол'
backup    lab07.sh '+Messages+'     play       work       Изображения  Шаблоны
conf.txt  lab07.sh~ my_os           project     Видео       Музыка
[edkrutova@fedora ~]$ touch backup.sh
[edkrutova@fedora ~]$ ls
abc1      conf.txt  lab07.sh~  my_os      project     Видео       Музыка
australia feathers  'main (14).cpp'  name_h     ski.places  Документы  Общедоступные
backup    file.txt  '#Messages+#'    name_h.txt text.txt    Загрузки   'Рабочий стол'
backup.sh lab07.sh '+Messages+'     play       work       Изображения  Шаблоны
[edkrutova@fedora ~]$
```

Рис. 4.2: Создание файла и директории



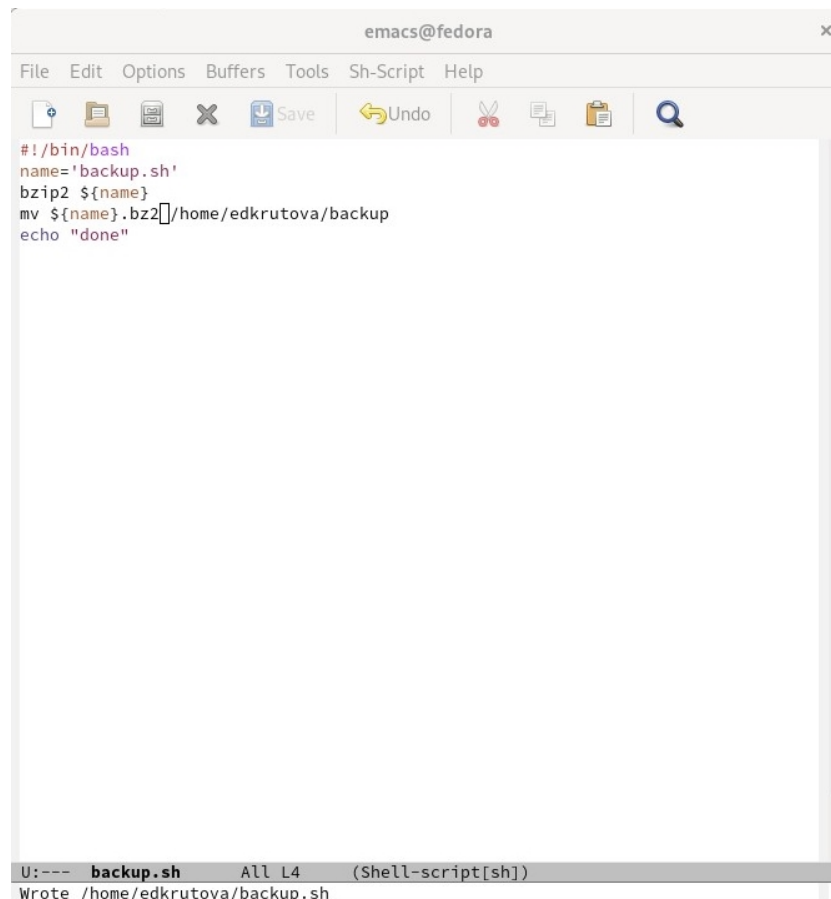


Рис. 4.3: Код программы



Рис. 4.4: Преобразование и исполнение

## 4.2 Задание 2 (рис. 4.5-4.7)

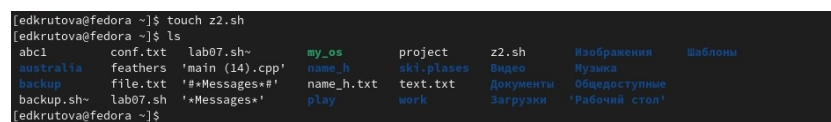


Рис. 4.5: Создание файла

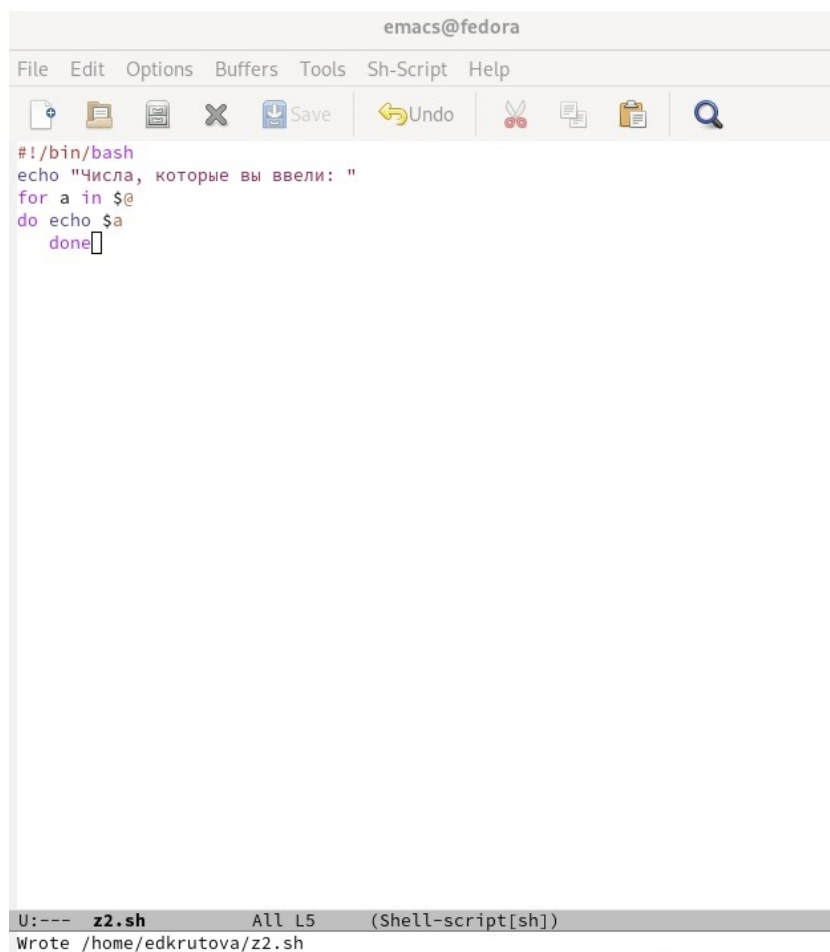
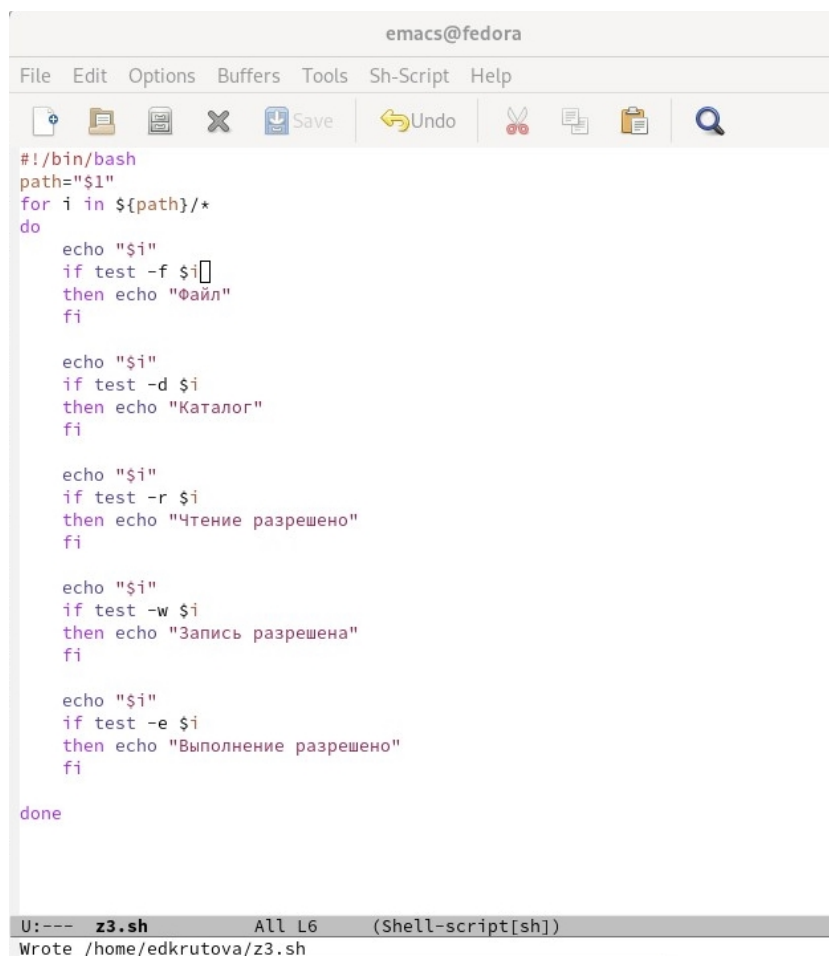


Рис. 4.6: Код программы



Рис. 4.7: Преобразование и исполнение

### 4.3 Задание 3 (рис. 4.8-4.9)



```
emacs@fedora
File Edit Options Buffers Tools Sh-Script Help
+ Save Undo
#!/bin/bash
path="$1"
for i in ${path}/*
do
    echo "$i"
    if test -f $i
    then echo "Файл"
    fi

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

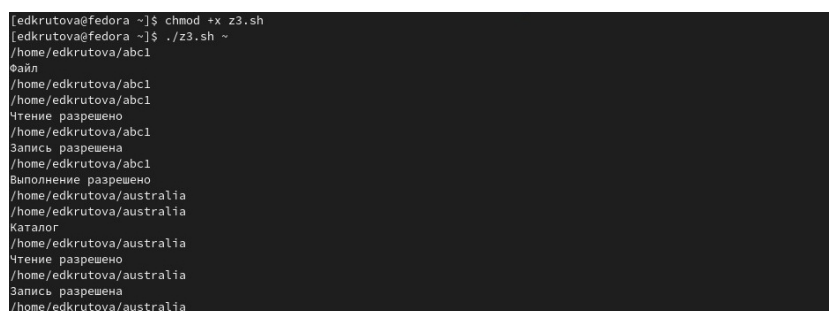
    echo "$i"
    if test -r $i
    then echo "Чтение разрешено"
    fi

    echo "$i"
    if test -w $i
    then echo "Запись разрешена"
    fi

    echo "$i"
    if test -e $i
    then echo "Выполнение разрешено"
    fi
done

U:--- z3.sh All L6 (Shell-script[sh])
Wrote /home/edkrutova/z3.sh
```

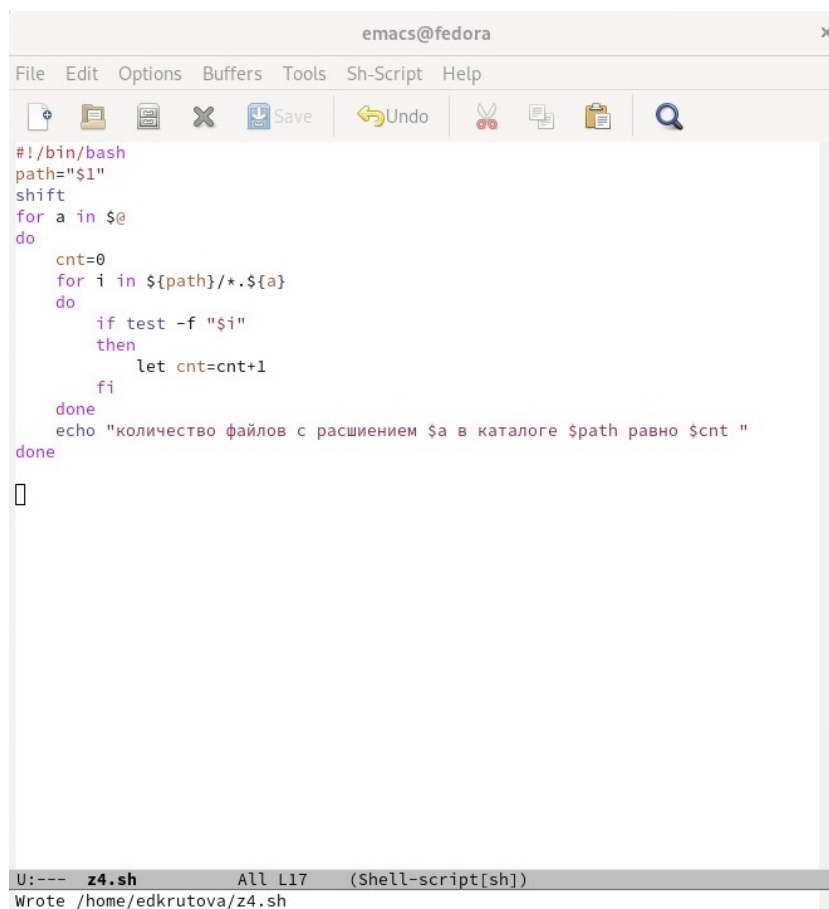
Рис. 4.8: Код программы



```
[edkrutova@fedora ~]$ chmod +x z3.sh
[edkrutova@fedora ~]$ ./z3.sh ~
/home/edkrutova/abc1
Файл
/home/edkrutova/abc1
/home/edkrutova/abc1
Чтение разрешено
/home/edkrutova/abc1
Запись разрешена
/home/edkrutova/abc1
Выполнение разрешено
/home/edkrutova/australia
/home/edkrutova/australia
Каталог
/home/edkrutova/australia
Чтение разрешено
/home/edkrutova/australia
Запись разрешена
/home/edkrutova/australia
```

Рис. 4.9: Преобразование и исполнение

## 4.4 Задание 4 (рис. 4.10-4.11)

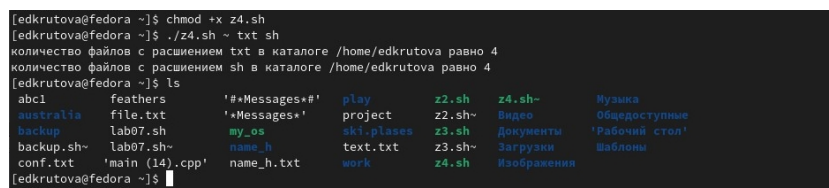


```
emacs@fedora
File Edit Options Buffers Tools Sh-Script Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]

#!/bin/bash
path="$1"
shift
for a in $@
do
    cnt=0
    for i in ${path}/*.${a}
    do
        if test -f "$i"
        then
            let cnt=cnt+1
        fi
    done
    echo "количество файлов с расшением $a в каталоге $path равно $cnt "
done
```

U:--- z4.sh All L17 (Shell-script[sh])  
Wrote /home/edkrutova/z4.sh

Рис. 4.10: Код программы



```
[edkrutova@fedora ~]$ chmod +x z4.sh
[edkrutova@fedora ~]$ ./z4.sh ~ txt sh
количество файлов с расшением txt в каталоге /home/edkrutova равно 4
количество файлов с расшением sh в каталоге /home/edkrutova равно 4
[edkrutova@fedora ~]$ ls
abcl feathers '*Messages*' play z2.sh z4.sh- Музыка
australia file.txt '*Messages*' project z2.sh- Видео Общедоступные
backup lab07.sh my_os ski.plases z3.sh- Документы 'Рабочий стол'
backup.sh- lab07.sh- name_h text.txt z3.sh- Загрузки Шаблоны
conf.txt 'main (14).cpp' name_h.txt work z4.sh- Изображения
```

Рис. 4.11: Преобразование и исполнение

## 5 Выводы

Я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать небольшие командные файлы.

## 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).

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.

## 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, введенные с клавиатуры, а переменная trash нужна для того, чтобы отобразить всю избыточно введенную 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 мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа , который, в свою очередь, является мета символом. 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'|'cd` 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` 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. `${#}` –возвращает целое число –количествослов, которые были результатом `$`;
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`);