

# **Лабораторная работа №10: отчет.**

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

Евдокимов Максим Михайлович. Группа - НФИбд-01-20.

# Содержание

<b>Цель работы</b>	<b>4</b>
<b>Задание</b>	<b>5</b>
<b>Указание к работе</b>	<b>6</b>
Описание метода . . . . .	6
<b>Выполнение лабораторной работы</b>	<b>8</b>
Задание 1 . . . . .	8
Код 1 . . . . .	9
Задание 2 . . . . .	9
Код 2 . . . . .	10
Задание 3 . . . . .	10
Код 3 . . . . .	11
Задание 4 . . . . .	12
Код 4 . . . . .	13
<b>Контрольные вопросы</b>	<b>14</b>
<b>Выводы</b>	<b>23</b>
<b>Список литературы</b>	<b>24</b>

# Список иллюстраций

1	Создание файла prog1 . . . . .	8
2	Код файла prog1 . . . . .	8
3	Результат запуска 1 . . . . .	9
4	Создание файла prog2 . . . . .	9
5	Код файла prog2 . . . . .	9
6	Результат запуска 2 . . . . .	10
7	Создание файла prog3 . . . . .	10
8	Код файла prog3 . . . . .	11
9	Результат запуска 3 . . . . .	11
10	Создание файла prog4 . . . . .	12
11	Код файла prog4 . . . . .	13
12	Результат запуска 4 . . . . .	13

# Цель работы

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

# Задание

1. Научится писать bash-скрипты.
2. Приобрести навыки по запуску и взаимодействию с bash-скриптами.

# Указание к работе

## Описание метода

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 оболочкой 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.

# Выполнение лабораторной работы

## Задание 1

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.

```
max@evdokimov:~$ mkdir lab10; cd lab10
bash: cd: команда не найдена
max@evdokimov:~127$ cd lab10
max@evdokimov:~/lab10$ mkdir backup; touch prog1.sh
max@evdokimov:~/lab10$ ls
backup  prog1.sh
max@evdokimov:~/lab10$ ls; ls -l backup; gedit prog1.sh
```

Рис. 1: Создание файла prog1



```
1 #!/bin/bash
2 tar -cvf ~/lab10/backup/backup.tar prog1.sh
```

Рис. 2: Код файла prog1



```

max@evdokimov:~/lab10$ gedit prog1.sh
(gedit:9247): Gtk-WARNING **: 06:14:11.409: Calling org.freedesktop.portal.Inhibit.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: Инте
рфейс «org.freedesktop.portal.Inhibit» для типа /org/freedesktop/portal/desktop объекта не найден
max@evdokimov:~/lab10$ bash prog1.sh: ls: -l backup
prog1.sh
backup prog1.sh
итого 12
-rw-r--r-- 1 max max 10240 дек 28 06:14 backup.tar
max@evdokimov:~/lab10$

```

Рис. 3: Результат запуска 1

## Код 1

```

#!/bin/bash
tar -cvf ~/lab10/backup/backup.tar prog1.sh

```

## Задание 2

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.

```

max@evdokimov:~/lab10$ touch prog2.sh
max@evdokimov:~/lab10$ gedit prog2.sh

```

Рис. 4: Создание файла prog2

```

Открыть  prog2.sh  Сохранить
~/lab10
1 #!/bin/bash
2 for A in $*
3     do echo $A
4 done

```

Рис. 5: Код файла prog2

```
max@evdokimov:~/lab10$ bash prog2.sh 1 2 3
1
2
3
max@evdokimov:~/lab10$ bash prog2.sh 3 2 1 0 10
3
2
1
0
10
max@evdokimov:~/lab10$
```

Рис. 6: Результат запуска 2

## Код 2

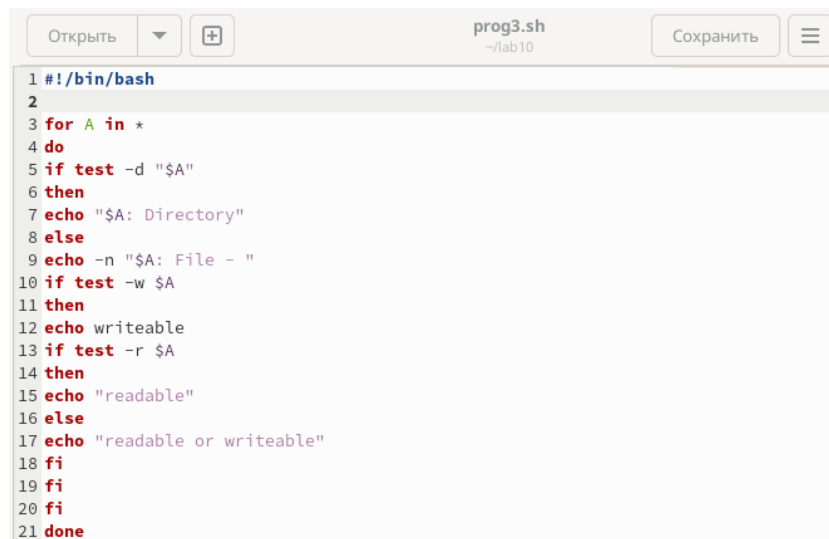
```
#!/bin/bash
for A in $*
do echo $A
done
```

## Задание 3

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.

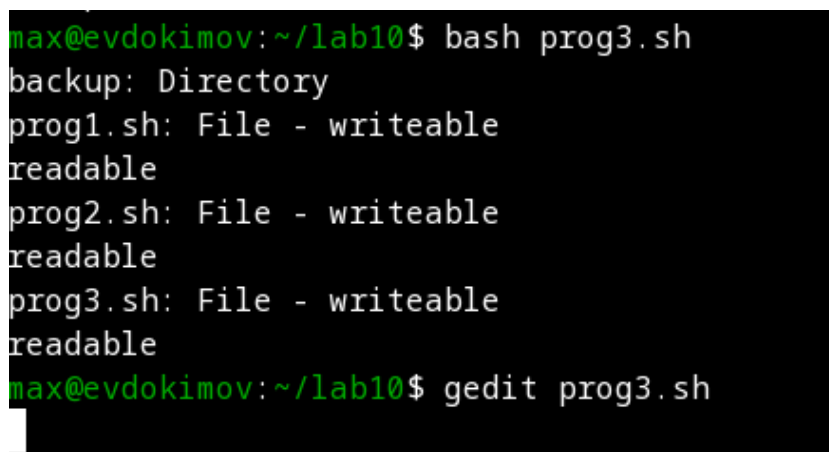
```
max@evdokimov:~/lab10$ touch prog3.sh
max@evdokimov:~/lab10$ gedit prog3.sh
```

Рис. 7: Создание файла prog3



```
1 #!/bin/bash
2
3 for A in *
4 do
5 if test -d "$A"
6 then
7 echo "$A: Directory"
8 else
9 echo -n "$A: File - "
10 if test -w $A
11 then
12 echo writeable
13 if test -r $A
14 then
15 echo "readable"
16 else
17 echo "readable or writeable"
18 fi
19 fi
20 fi
21 done
```

Рис. 8: Код файла prog3



```
max@evdokimov:~/lab10$ bash prog3.sh
backup: Directory
prog1.sh: File - writeable
readable
prog2.sh: File - writeable
readable
prog3.sh: File - writeable
readable
max@evdokimov:~/lab10$ gedit prog3.sh
```

Рис. 9: Результат запуска 3

## Код 3

```
#!/bin/bash
for A in *
do
if test -d "$A"
```

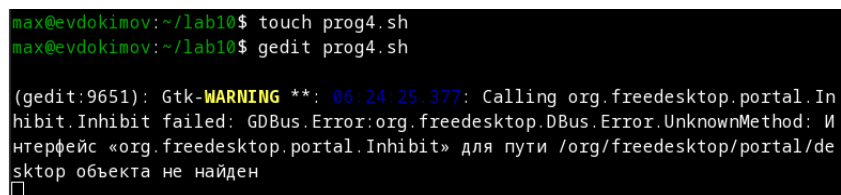
```

then
    echo "$A: Directory"
else
    echo -n "$A: File - "
    if test -w $A
    then
        echo writeable
    if test -r $A
    then
        echo "readable"
    else
        echo "readable or writeable"
    fi
    fi
fi
done

```

## Задание 4

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.



```

max@evdokimov:~/lab10$ touch prog4.sh
max@evdokimov:~/lab10$ gedit prog4.sh

(gedit:9651): Gtk-WARNING **: 06:24:25.377: Calling org.freedesktop.portal.Inhibit.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: И
нтерфейс «org.freedesktop.portal.Inhibit» для пути /org/freedesktop/portal/de
sktop объекта не найден

```

Рис. 10: Создание файла prog4

Рис. 11: Код файла prog4

Рис. 12: Результат запуска 4

## Код 4

```
#!/bin/bash
format=""
directory=""
echo "Укажите формат: "
read format
echo "Напишите директорию: "
read directory
find "${directory}" -name "*${format}" -type f | wc -l
```

# Контрольные вопросы

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

Командные оболочки - это программы (процессы), 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 shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой 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).

2. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) - интерфейс переносимой операционной системы для компьютерных сред. Пред

ставляет собой набор стандартов, подготовленных институтом инженеров по электронике и радиотехнике (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 мета символ.

### 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`?

- “!” `!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`;
- “\*” *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`;
- “-” *-expr* Операция отрицания `expr` (называется унарный минус) `- expr1 - expr2` Вычитает `expr2` из `expr1` `-- var -- expr` вычитает `expr` из значения `var` и присваивает результат `var`;
- “/” `expr / expr2` 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 `<< expr1 << expr2` Сдвигает `expr1` влево на `expr2` бит;
- “<=” `var <= expr` Побитовый сдвиг влево значения `var` на `expr` `<= expr1 <= expr2` Если 16 `expr1` меньше, или равно `expr2`, возвращает 1; иначе возвращает 0;



- “=” var = exp Присваивает значение exp переменной var;
- “==” exp1==exp2 Если exp1 равно exp2. Возвращает 1; иначе возвращает 0;
- “>” exp1 > exp2 1 если exp1 больше, чем exp2; иначе 0;
- “>=” exp1 >= exp2 1 если exp1 больше, или равно exp2; иначе 0 » exp » exp2  
Сдвигает exp1 вправо на exp2 бит »= var »=exp Побитовый сдвиг вправо значения var на exp;
- “^” exp1 ^ exp2 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 `Sh` имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды `set`.

#### 8. Что такое метасимволы?

Такие символы, как `"`, `<`, `>`, `*`, `?`, `|`, `"`, `"`, `&` являются мета-символами 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=Alaska`. Индексация массивов начинается с нулевого элемента. В командном процессоре `Cи` 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. 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 терминале будет выведена строка, содержащая номер терминала, используемого указанным пользователем.

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; 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 были изучены и повторены методы и принципы написания и запуска bash-скриптов.

# Список литературы

1. Лабораторная работа №10
2. Основы bash скриптов
3. Написание bash скриптов