

# **Отчёт по лабораторной работе №10**

**дисциплина: Операционные системы**

Студент: Коннова Татьяна Алексеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>13</b>
<b>5</b>	<b>Выводы</b>	<b>20</b>

## Список иллюстраций

3.1	Создание нового каталога и файла для скрипта . . . . .	7
3.2	Написание первого скрипта . . . . .	8
3.3	Право на выполнение, запуск файла и проверка . . . . .	8
3.4	Создание второго файла и открытие emacs . . . . .	9
3.5	Написание второго скрипта . . . . .	9
3.6	Право на выполнение, запуск файла . . . . .	10
3.7	Создание третьего файла . . . . .	10
3.8	Написание третьего скрипта . . . . .	10
3.9	Право на выполнение, запуск файла для каталога backup . . . . .	11
3.10	Создание четвёртого файла . . . . .	11
3.11	Написание четвёртого скрипта . . . . .	12
3.12	Право на выполнение, запуск файла для форматов .txt и .pdf . . .	12

## Список таблиц

# 1 Цель работы

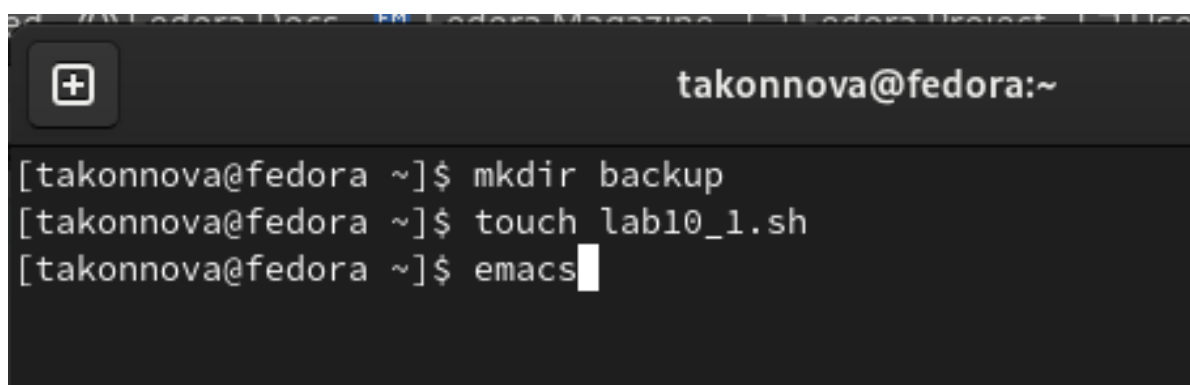
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

## 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 Выполнение лабораторной работы

Откроем терминал и создадим в домашнем каталоге папку backup. После чего создадим файл lab10\_1.sh для написания скрипта. Откроем emacs (рис. [3.1]).

A screenshot of a terminal window with a dark background. The title bar at the top shows a window icon and the text 'takonnova@fedora:~'. The terminal content shows three lines of commands and their prompts: '[takonnova@fedora ~]\$ mkdir backup', '[takonnova@fedora ~]\$ touch lab10\_1.sh', and '[takonnova@fedora ~]\$ emacs' followed by a white cursor block.

```
takonnova@fedora:~  
[takonnova@fedora ~]$ mkdir backup  
[takonnova@fedora ~]$ touch lab10_1.sh  
[takonnova@fedora ~]$ emacs
```

Рис. 3.1: Создание нового каталога и файла для скрипта

В emacs откроем созданный файл lab10\_1.sh и приступим к написанию скрипта, 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. [3.2]).

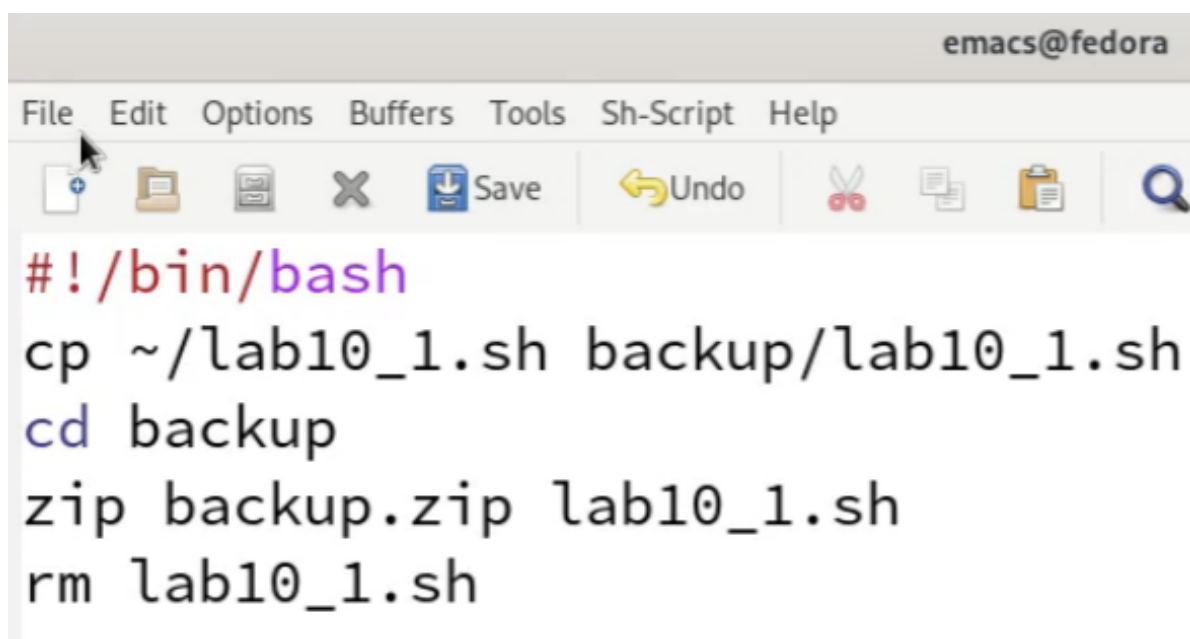


Рис. 3.2: Написание первого скрипта

После того как скрипт написан мы сохраняем файл и закрываем emacs. В терминале мы даём этому файлу право на выполнение. Теперь запустим этот файл и перейдём в каталог backup для проверки командой ls (рис. [3.3]).

```
[takonnova@fedora ~]$ chmod u+x lab10_1.sh
[takonnova@fedora ~]$ ./lab10_1.sh
[takonnova@fedora ~]$ cd backup
```

Рис. 3.3: Право на выполнение, запуск файла и проверка

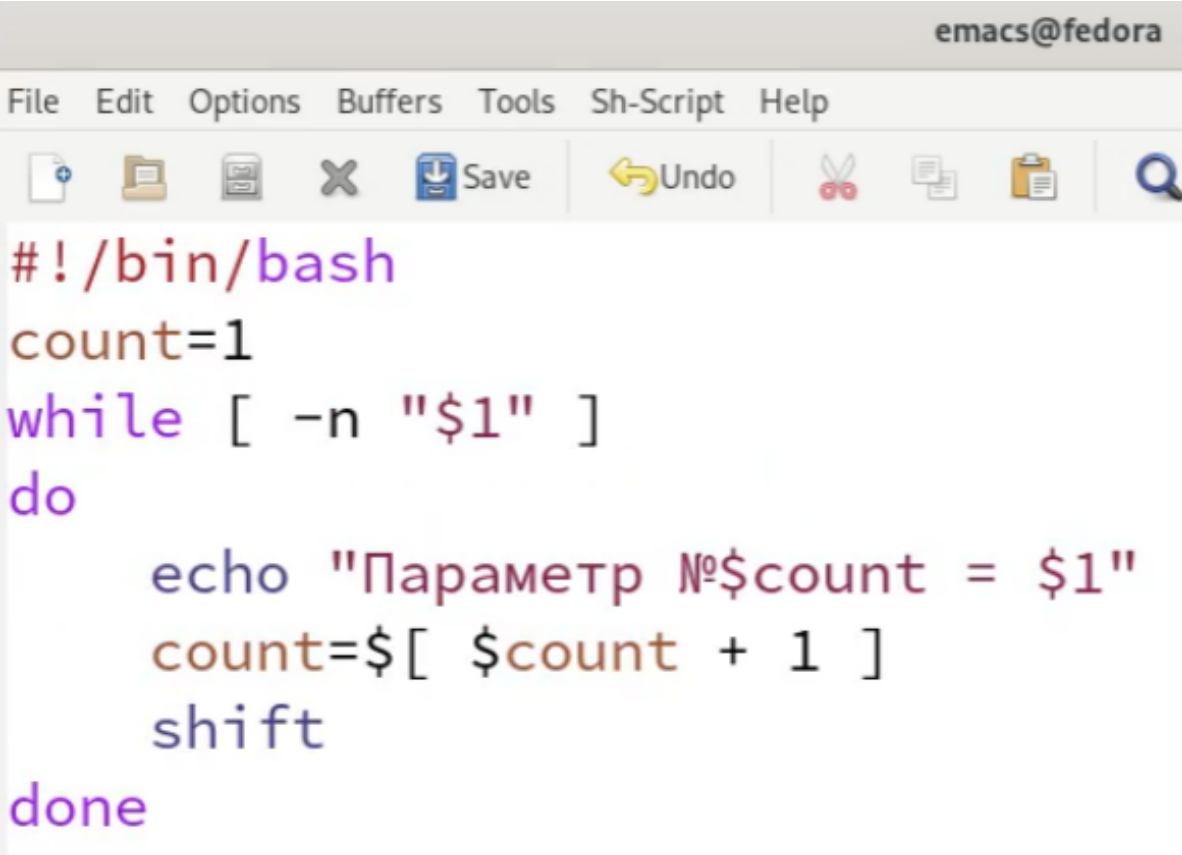
Возвращаемся в домашний каталог и создаём второй файл для скрипта lab10\_2.sh. Открываем emacs (рис. [3.4]).



```
[takonnova@fedora backup]$ touch lab10_2.sh  
[takonnova@fedora backup]$ emacs
```

Рис. 3.4: Создание второго файла и открытие emacs

Открываем файл lab10\_2.sh и начинаем писать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт может последовательно распечатывать значения всех переданных аргументов.(рис. [3.5]).

The image shows a screenshot of the Emacs text editor window. The title bar at the top right says 'emacs@fedora'. Below the title bar is a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Under the menu bar is a toolbar with icons for file operations (new, open, save, close, save all), editing (undo, redo, cut, copy, paste), and search. The main text area contains a shell script with syntax highlighting: the shebang '#!/bin/bash' is in red and blue; 'count=1' is in brown; 'while' and 'done' are in purple; '[' and ']' are in blue; '-n' is in red; '\$1' is in red; 'echo' is in blue; 'Параметр №\$count = \$1' is in red; '\$count' is in brown; '+ 1' is in blue; and 'shift' is in blue. The script is as follows:

```
#!/bin/bash  
count=1  
while [ -n "$1" ]  
do  
    echo "Параметр №$count = $1"  
    count=$((count + 1))  
    shift  
done
```

Рис. 3.5: Написание второго скрипта

Сохраняем файл и также даём в терминале право на выполнение. Запускаем файл lab10\_2.sh (рис. [3.6]).

```
[takonnova@fedora backup]$ chmod u+x lab10_2.sh
[takonnova@fedora backup]$ ./lab10_2.sh blum tecna stella leila muza flora roxy
[takonnova@fedora backup]$
```

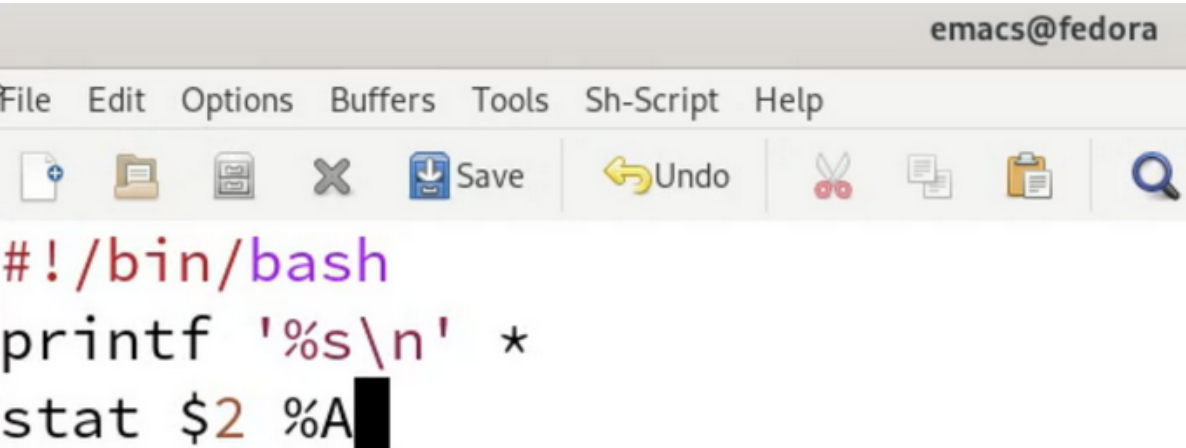
Рис. 3.6: Право на выполнение, запуск файла

Снова переходим в домашний каталог и создаём третий файл. Запускаем emacs (рис. [3.7]).

```
[takonnova@fedora ~]$ touch lab10_3.sh
[takonnova@fedora ~]$ emacs
```

Рис. 3.7: Создание третьего файла

После открытия файла lab10\_3.sh напишем командный файл — аналог команды ls (без использования самой этой команды и команды dir). В котором требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога. (рис. [3.8]).

The image shows a screenshot of the Emacs text editor window. The title bar at the top right says "emacs@fedora". Below the title bar is a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Under the menu bar is a toolbar with icons for creating a new file, opening a file, saving a file, closing a file, saving, undo, redo, copy, paste, and search. The main text area contains the following code:

```
#!/bin/bash
printf '%s\n' *
stat $2 %A
```

Рис. 3.8: Написание третьего скрипта

Сохраняем наш скрипт и даём право на выполнение. Запускаем файл для каталога backup (рис. [3.9]).

```
#abc1#
abc1
backup
bin
#lab07.sh#
lab07.sh
lab10_1.sh
lab10_1.sh~
lab10_2.sh
lab10_2.sh~
lab10_3.sh
lab10_3.sh~
work
Видео
Документы
Загрузки
Изображения
Музыка
Общедоступные
Рабочий стол
Шаблоны
  Файл: backup
  Размер: 20          Блоков: 0          Блок В/В: 4096   каталог
Устройство: 26h/38d  Инода: 81523       Ссылки: 1
```

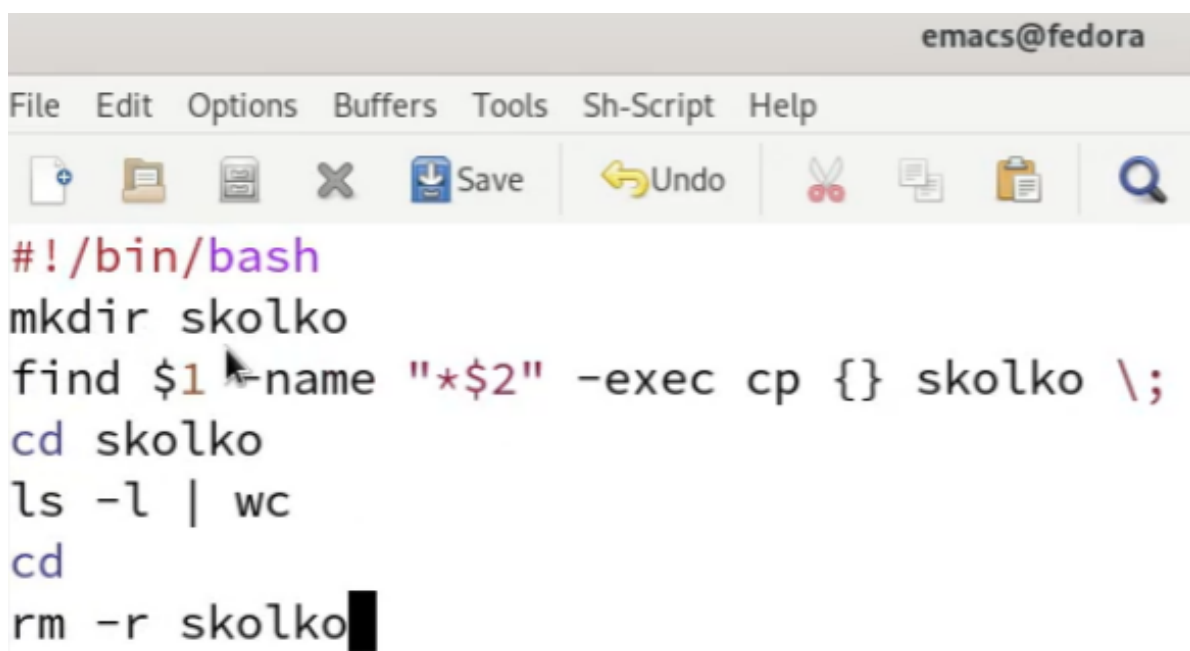
Рис. 3.9: Право на выполнение, запуск файла для каталога backup

Переходим в домашний каталог и создаём последний файл для четвёртого скрипта. Запускаем emacs (рис. [3.10]).

```
[takonnova@fedora ~]$ touch lab10_4.sh
[takonnova@fedora ~]$ emacs
```

Рис. 3.10: Создание четвёртого файла

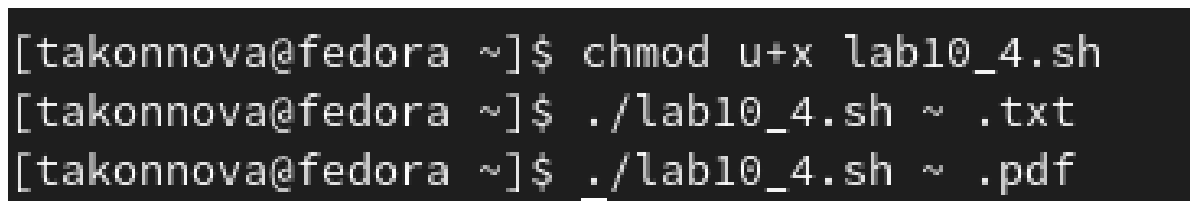
В четвёртом файле напишем командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки. (рис. [3.11]).

The image shows the Emacs editor interface on a Fedora system. The title bar at the top reads 'emacs@fedora'. Below it is a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. A toolbar contains icons for file operations like 'Save', 'Undo', 'Cut', 'Copy', and 'Paste'. The main text area contains a shell script with the following lines: 

```
#!/bin/bash
mkdir skolko
find $1 -name "$2" -exec cp {} skolko \;
cd skolko
ls -l | wc
cd
rm -r skolko
```

Рис. 3.11: Написание четвёртого скрипта

Сохраним файл и выйдем из emacs. Как делали ранее, дадим файлу право на выполнение и запустим его для двух форматов: .txt и .pdf (рис. [3.12]).

The image shows a terminal window with the following commands and output: 

```
[takonnova@fedora ~]$ chmod u+x lab10_4.sh
[takonnova@fedora ~]$ ./lab10_4.sh ~ .txt
[takonnova@fedora ~]$ ./lab10_4.sh ~ .pdf
```

Рис. 3.12: Право на выполнение, запуск файла для форматов .txt и .pdf

## 4 Контрольные вопросы

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

Командный процессор это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: 1. оболочка Борна (Bourne shell или sh) это стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; 2. C оболочка (или csh) это надстройка на оболочкой Борна, использующая Подобный синтаксис команд с возможностью сохранения истории выполнения команд; 3. Оболочка Корна (или ksh) 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).

2. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments ) это 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`?

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. 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 month day trash`». В переменные `month` 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 (( )) можно записывать условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7. Какие стандартные имена переменных Вам известны?

Стандартные переменные:

- `PATH`: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной `PATH`, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
- `PS1` и `PS2`: эти переменные предназначены для отображения промптера командного процессора. `PS1` это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер `PS2`. Он по умолчанию имеет значение символа >.
- `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 (`newline`).

- 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 Youhavemail(y Вас есть почта).
- TERM: тип используемого терминала.
- LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

#### 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 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. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществить её 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, 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?

Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами `test -f [путь до файла]` (для проверки, является ли обычным файлом) и `test -d [путь до файла]` (для проверки, является ли каталогом).

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

Команду `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. 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 неё 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 неё имени данного командного файла.

15. Назовите специальные переменные языка bash и их назначение.

Специальные переменные: - \$\* 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; - \$# возвращает целое число количеств слов, которые были результатом \$; - \${#name} возвращает целое значение длины строки в переменной name; - \${name[n]} обращение к n му элементу массива; - \${name[\*]} перечисляет все элементы массива, разделённые 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 удалённым самым коротким левым образцом

(pattern); - `${#name[*]}` и `${#name[@]}` эти выражения возвращают количество элементов в массиве name.

## **5 Выводы**

В ходе выполнения лабораторной работы мы изучили основы программирования в оболочке ОС UNIX/Linux и научились писать небольшие командные файлы.