

# **Отчёт по лабораторной работе №11**

**дисциплина: Операционные системы**

Быстров Глеб Андреевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Теория</b>	<b>4</b>
<b>3</b>	<b>Задание</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>14</b>
<b>6</b>	<b>Выводы</b>	<b>21</b>
<b>7</b>	<b>Библиографический список</b>	<b>22</b>

# 1 Цель работы

В данной лабораторной работе мне будет необходимо изучить основы программирования в оболочке ОС UNIX/Linux. Будет необходимо научиться писать небольшие командные файлы.

## 2 Теория

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.

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.

При перечислении имён файлов текущего каталога можно использовать следующие символы: `*` — соответствует произвольной, в том числе и пустой строке; `?` — соответствует любому одинарному символу; `[c1-c1]` — соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. Например, `echo *` — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls.c` — *выведет все файлы с последними двумя символами, совпадающими с c.* `echo prog.?` — *выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog.* `[a-z]` — соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `bash командный_файл [аргументы]`

Весьма необходимой при программировании является команда `getopts`, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg ...]`

В обобщённой форме оператор цикла `for` выглядит следующим образом: `for имя [in список-значений] do список-команд done`

Оператор выбора `case` реализует возможность ветвления на произвольное число ветвей. Эта возможность обеспечивается в большинстве современных языков программирования, предполагающих использование структурного подхода. В обобщённой форме оператор выбора `case` выглядит следующим образом: `case имя in шаблон1) список-команд;; шаблон2) список-команд;; ... esac`

В обобщённой форме условный оператор `if` выглядит следующим образом:

```
if список-команд then список-команд {elif список-команд then список-команд}  
[else список-команд] fi
```

В обобщённой форме оператор цикла `while` выглядит следующим образом: Кулябов Д. С. и др. Операционные системы 63 `while список-команд do список-команд done`

Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов.

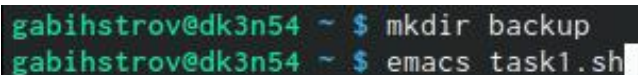
## 3 Задание

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.

## 4 Выполнение лабораторной работы

- ЗАДАНИЕ 1

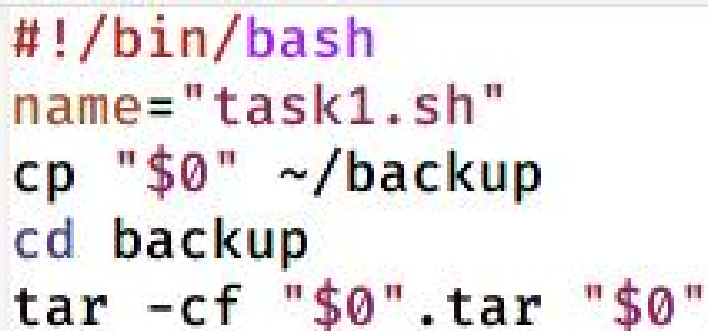
1. Создал директорию в домашнем каталоге с помощью команды `mkdir`. С помощью команды `emacs` создал файл `task1.sh`. (рис. 4.1)



```
gabihstrov@dk3n54 ~ $ mkdir backup
gabihstrov@dk3n54 ~ $ emacs task1.sh
```

Figure 4.1: Создание директории и файла

2. Скрипт, который при запуске делает резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию `backup` в вашем домашнем каталоге. Файл архивируется архиватором `tar`. В начале мы вызываем `bash`. Затем создаём переменную `name` в которую записываем название файла. Используем команду `cp` для копирования файла в директорию `backup`. Далее переходим в каталог и создаём архив. (рис. 4.2)

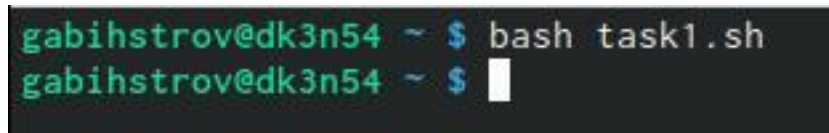


```
#!/bin/bash
name="task1.sh"
cp "$0" ~/backup
cd backup
tar -cf "$0".tar "$0"
```

Figure 4.2: Скрипт



3. Команда для запуска скрипта. (рис. 4.3)



```
gabihstrov@dk3n54 ~ $ bash task1.sh
gabihstrov@dk3n54 ~ $
```

Figure 4.3: Запуск скрипта

4. Скрипт успешно создал резервную копию и архив. (рис. 4.4)

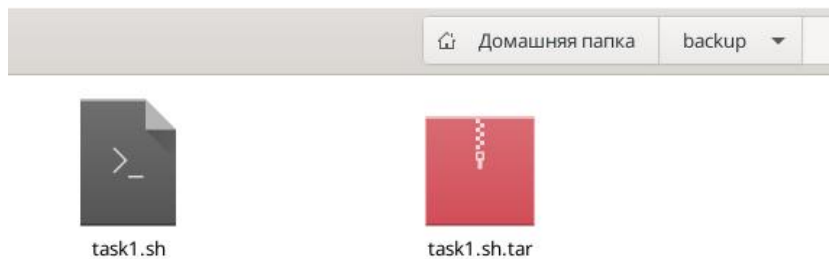
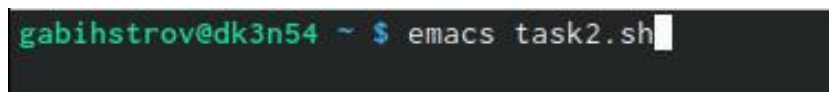


Figure 4.4: Демонстрация работы скрипта

- ЗАДАНИЕ 2

5. С помощью команды emacs создал файл task2.sh. (рис. 4.5)



```
gabihstrov@dk3n54 ~ $ emacs task2.sh
```

Figure 4.5: Создание файла

6. Скрипт командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт последовательно распечатывает значения всех переданных аргументов. В начале мы вызываем `bash`. Далее вызываем массив `mas`. Выводим надпись для запроса данных. Считываем введённые данные. Выводим элементы массива. (рис. 4.6)

```
#!/bin/bash
declare -a mas
echo "Введите аргументы"
read -a mas
echo ${mas[@]}
```

Figure 4.6: Скрипт

7. Командный файл успешно распечатал значения переданных аргументов.  
(рис. 4.7)

```
gabihstrov@dk3n54 ~ $ bash task2.sh
Введите аргументы
1 3 5 7 9 11 13
1 3 5 7 9 11 13
gabihstrov@dk3n54 ~ $
```

Figure 4.7: Демонстрация работы

- ЗАДАНИЕ 3

8. С помощью команды emacs создал файл task3.sh. (рис. 4.8)

```
gabihstrov@dk3n54 ~ $ emacs task2.sh
```

Figure 4.8: Создание файла

9. Написал командный файл — аналог команды ls (без использования самой этой команды и команды dir). Скрипт выдаёт информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого

каталога. В начале мы вызываем `bash`. Выводим надпись для запроса названия каталога. Считываем введенные данные. Переходим в необходимый каталог. Выводим содержимое каталога. Ключ `-c` выводит файлы построчно. `%A` выводит права доступа, а `%n` выводит названия файлов. (рис. 4.9)

```
#!/bin/bash
echo "Введите путь к каталогу:"
read name
echo "Имя: $name"
cd ${name}
echo "Файлы и права доступа:"
stat -c '%A %n' *
```

Figure 4.9: Скрипт

10. Командный файл успешно вывел названия файлов и права доступа введенного каталога. (рис. 4.10)

```
gabihstrov@dk3n54 ~ $ bash task3.sh
Введите путь к каталогу:
monthly
Имя: monthly
Файлы и права доступа:
-rw-r--r-- april
-rw-r--r-- june
-rw-r--r-- may
gabihstrov@dk3n54 ~ $
```

Figure 4.10: Демонстрация работы

- ЗАДАНИЕ 4

11. С помощью команды `emacs` создал файл `task4.sh`. (рис. 4.11)

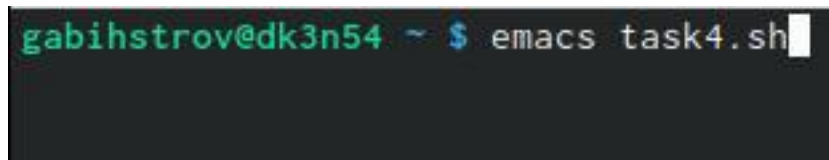


Figure 4.11: Создание файла

12. Написал 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. В начале мы вызываем bash. Объявляем переменную directory и считываем название директории. Объявляем переменную format и считываем необходимый формат файла для поиска. Затем переходим в заданную директорию и находим количество файлов с нужным форматом. (рис. 4.12)

```
#!/bin/bash
directory=""
echo "Введите название директории"
read directory
format=""
echo "Введите формат для поиска"
read format
cd ${directory}
find -name ".*$format" | wc -l
```

Figure 4.12: Скрипт

13. Командный файл успешно вывел количество файлов с заданным форматом в необходимой директории. (рис. 4.13)

```
gabihstrov@dk3n54 ~ $ bash task4.sh
Введите название директории
backup
Введите формат для поиска
tar
1
gabihstrov@dk3n54 ~ $
```

Figure 4.13: Демонстрация работы

## 5 Контрольные вопросы

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

*Библиографический список ссылка №1*

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).

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 ${mark}` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}`. Например, использование команд `b=/tmp/andy-ls -l myfile > bl` *`ssudo apt — getinstalltexlive — luatexls/tmp/andy — ls, ls — l`* `>bls` приведёт к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то её значением будет символ пробела. 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 `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 отобрать всю избыточно введенную информацию и игнорировать её. Изъять переменную из программы можно с помощью команды `unset`.

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

Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).

#### 6. Что означает операция (( ))?

Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки — (( )) .

#### 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. В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set .

#### 8. Что такое метасимволы?

##### *Библиографический список ссылка №2*

Такие символы, как ' < > \* ? | " & , являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола.

#### 9. Как экранировать метасимволы?

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 \* , - echo ab'|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

#### 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`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определённые на текущий момент функции; `-ft` — при следующем вызове функции инициализирует её 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 одноимёнными именами функций, загружает его и вызывает эти функции.

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

Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения. Так например, команда `test -f file60` возвращает нулевой код завершения (истина), если файл `file` существует, и ненулевой код завершения (ложь) в противном случае: `-test s` — истина, если аргумент `s` имеет значение истина; `-test -f file` — истина, если файл `file` существует; `-test -i file` — истина, если файл `file` доступен по чтению; `-test -w file` — истина, если файл `file` доступен по записи; `-test -e file` — истина, если файл `file` — исполняемая программа; `-test -d file` — истина, если файл `file` является каталогом.

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

- `set` 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.

- typeset

Если использовать typeset -i для объявления и присвоения переменной, то при последующем её применении она станет целой. Также можно использовать ключевое слово integer (псевдоним для typeset -i ) и объявлять таким образом переменные целыми. Выражения типа  $x=y+z$  будет восприниматься в это случае как арифметические.

Команда typeset имеет четыре опции для работы с функциями: – -f — перечисляет определённые на текущий момент функции; – -ft — при последующем вызове функции иницирует её 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 одноимёнными именами функций, загружает его и вызывает эти функции.

- unset

Изъять переменную из программы можно с помощью команды unset . Удалить функцию можно с помощью команды unset с флагом -f .

#### 14. Как передаются параметры в командные файлы?

##### *Библиографический список ссылка №3*

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. Рассмотрим это на примере.

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

Вот ещё несколько специальных переменных, используемых в командных файлах: -  $\$*$  — 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  $n$ -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  $name$  .

## 6 Выводы

В данной лабораторной работе мне успешно удалось изучить основы программирования в оболочке ОС UNIX/Linux. Получилось научиться писать небольшие командные файлы.

## 7 Библиографический список

1. О разных командных оболочках Linux и Unix (<https://habr.com/ru/post/157283/>)
2. Метасимволы в регулярных выражениях (<https://housecomputer.ru/programming/regexp/m>)
3. Аргументы командной строки Bash (<https://losst.ru/argumenty-komandnoj-stroki-bash>)