

# **Отчёт по лабораторной работе №10**

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

Хусаинова Динара Айратовна

# Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Ход работы	8
4	Контрольные вопросы	14
5	Вывод	19

## Список иллюстраций

3.1	Командный код . . . . .	8
3.2	Запускаем командный файл . . . . .	9
3.3	Проверяем работу . . . . .	9
3.4	Создание файла для второго скрипта . . . . .	9
3.5	Проверка работы файла . . . . .	10
3.6	Создаем файл . . . . .	10
3.7	Пишем команды . . . . .	11
3.8	Создаем исполняемый файл и запускаем . . . . .	11
3.9	Наблюдаем вывод информации о файлах и каталогах . . . . .	12
3.10	Создание файла для 4 скрипта . . . . .	12
3.11	Вводим команды в файл . . . . .	13
3.12	Запускаем . . . . .	13

## List of Tables

# 1 Цель работы

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

## 2 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:

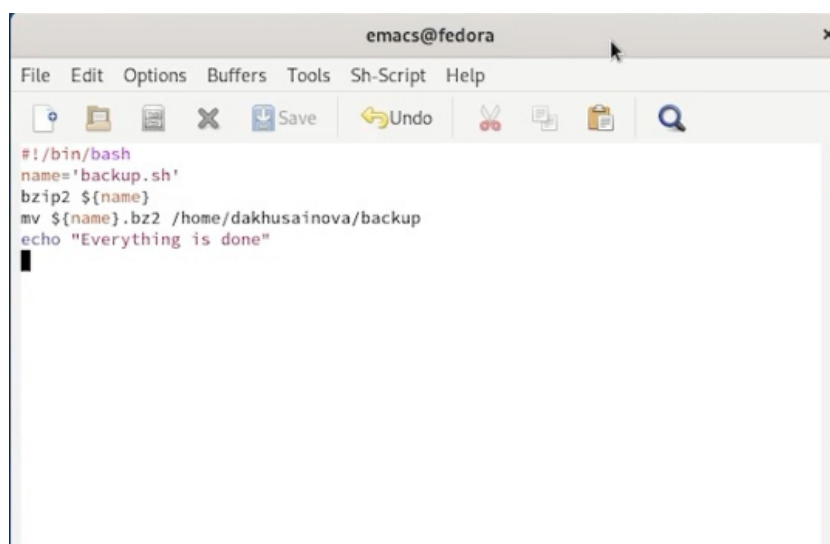
- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (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. 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.

## 3 Ход работы

1. Напишем первый наш скрипт, 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 будет архивироваться архиватором bzip2(рис. 3.1). После этого создаем нужную директорию, а после делаем файл исполняемым, запускаем(рис. 3.2), проверяем работу (рис. 3.3).

The image shows a screenshot of an Emacs editor window titled 'emacs@fedora'. The window has a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu bar is a toolbar with icons for file operations like 'Save', 'Undo', 'Cut', 'Copy', 'Paste', and 'Find'. The main text area contains a shell script with the following content:

```
#!/bin/bash
name='backup.sh'
bzip2 ${name}
mv ${name}.bz2 /home/dakhusainova/backup
echo "Everything is done"
```

A cursor is visible at the end of the last line of the script.

Рис. 3.1: Командный код



```
[dakhusainova@fedora ~]$ mkdir backup
[dakhusainova@fedora ~]$ ls
abcl          feathers    july        reports     'Без имени 1'
australia     file        '#lab07.sh#' rooot.txt   Видео
backup        file1.sh    lab07.sh    rudo.txt    Документы
backup.sh     file2.sh    lab07.sh~   ski.places  Загрузки
backup.sh~    file3.sh    may         temp        Изображения
bill          file4       monthly     text1.txt   Музыка
bin           file.txt    my_os       text.cpp     Общедоступные
conf.txt      go.txt      non.txt     text.txt    'Рабочий стол'
dakhusainova hot.txt     play        work        Шаблоны
[dakhusainova@fedora ~]$ chmod +x backup.sh
[dakhusainova@fedora ~]$ ./backup.sh
Everything is done
[dakhusainova@fedora ~]$
```

Рис. 3.2: Запускаем командный файл

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

Рис. 3.3: Проверяем работу

2. Теперь напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Выведем последовательно все введенные числа. Создаем файл, открываем редактор(рис. 3.4), пишем код и проверяем работу, предварительно создав исполняемый файл( рис. 3.5).

```
[dakhusainova@fedora ~]$ touch task2.sh
[dakhusainova@fedora ~]$ emacs &
```

Рис. 3.4: Создание файла для второго скрипта

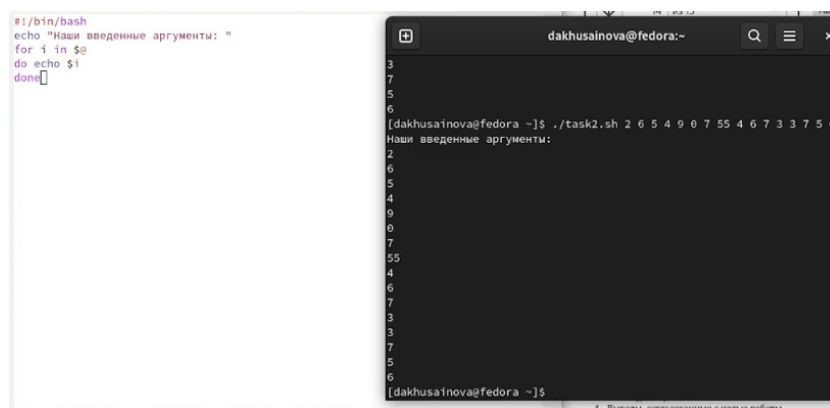


Рис. 3.5: Проверка работы файла

3. Напишем командный файл — аналог команды `ls` без использования самой этой команды и команды `dir`. От нас требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога. Создаем также файл, открываем редактор, пишем команды, а потом проверяем (рис. 3.6,3.7,3.8,3.9).

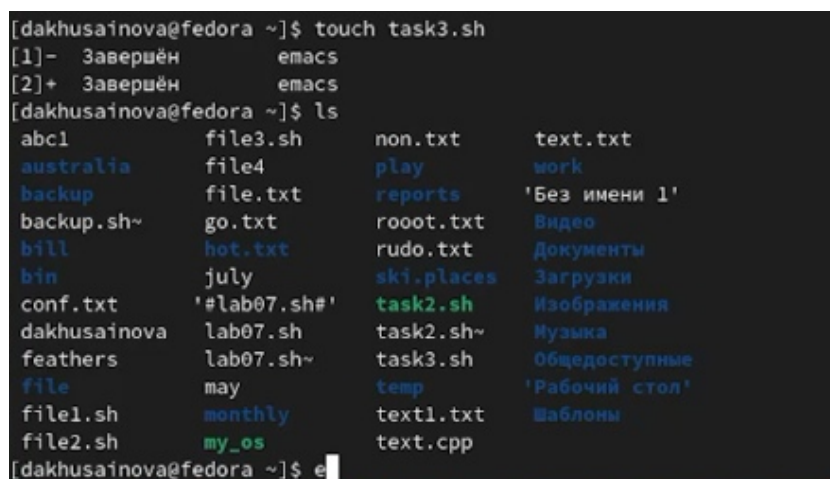
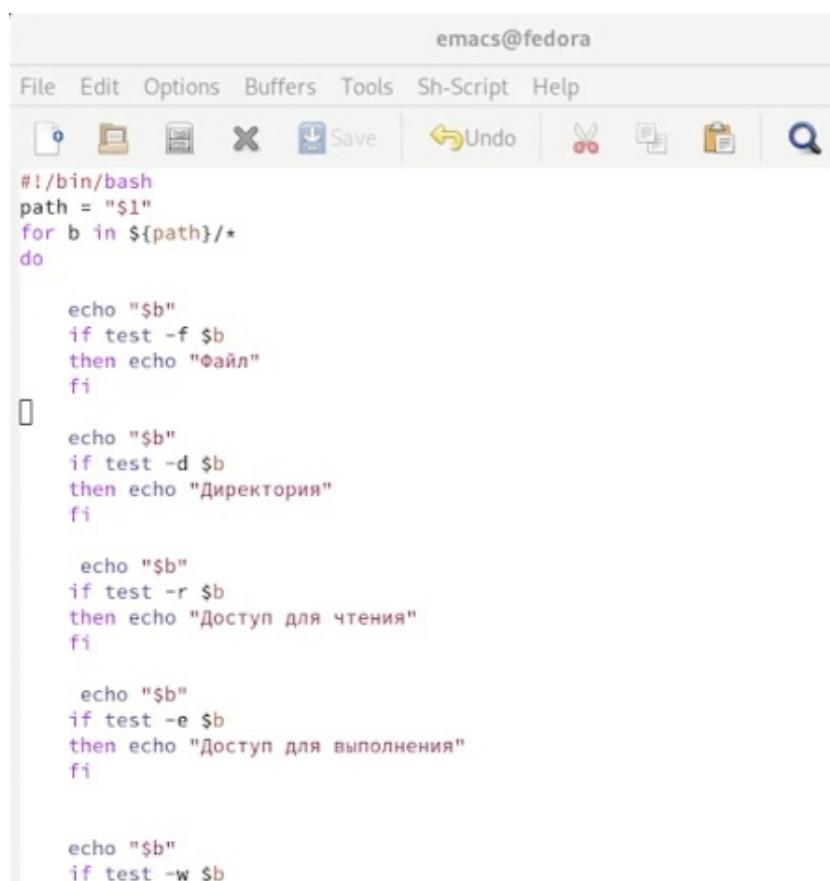


Рис. 3.6: Создаем файл

The image shows the Emacs editor interface on a Fedora system. The title bar reads 'emacs@fedora'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. Below the menu is a toolbar with icons for opening files, saving, undo, redo, and search. The main text area contains a shell script in Bash. The script starts with a shebang line, sets a path variable, and uses a 'for' loop to iterate over files and directories in the specified path. For each item, it checks if it's a file, directory, readable, executable, or writable, and prints a message in Russian. The script is currently at the end of the loop, with the cursor on the 'do' line.

```
#!/bin/bash
path = "$1"
for b in ${path}/*
do

    echo "$b"
    if test -f $b
    then echo "Файл"
    fi

    echo "$b"
    if test -d $b
    then echo "Директория"
    fi

    echo "$b"
    if test -r $b
    then echo "Доступ для чтения"
    fi

    echo "$b"
    if test -e $b
    then echo "Доступ для выполнения"
    fi

    echo "$b"
    if test -w $b
```

Рис. 3.7: Пишем команды

The image shows a terminal window with the prompt '[dakhusainova@fedora ~]\$'. The user has entered the command './task3.sh /home/dakhusainova', which is highlighted in blue.

```
[dakhusainova@fedora ~]$ ./task3.sh /home/dakhusainova
```

Рис. 3.8: Создаем исполняемый файл и запускаем

```

/tmp
Доступ для чтения
/tmp
Доступ для выполнения
/tmp
Доступ для записи
/usr
/usr
Директория
/usr
Доступ для чтения
/usr
Доступ для выполнения
/usr
/var
/var
Директория
/var
Доступ для чтения
/var
Доступ для выполнения
/var

```

Рис. 3.9: Наблюдаем вывод информации о файлах и каталогах

4. Пишем 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. Создаем файл, пишем код и запускаем, видим, сколько файлов заданного формата есть в проверяемом каталоге(рис. 3.10,3.11,3.12).

```

[dakhusainova@fedora ~]$ touch task4.sh
[1]+  Завершён      emacs
[dakhusainova@fedora ~]$

```

Рис. 3.10: Создание файла для 4 скрипта

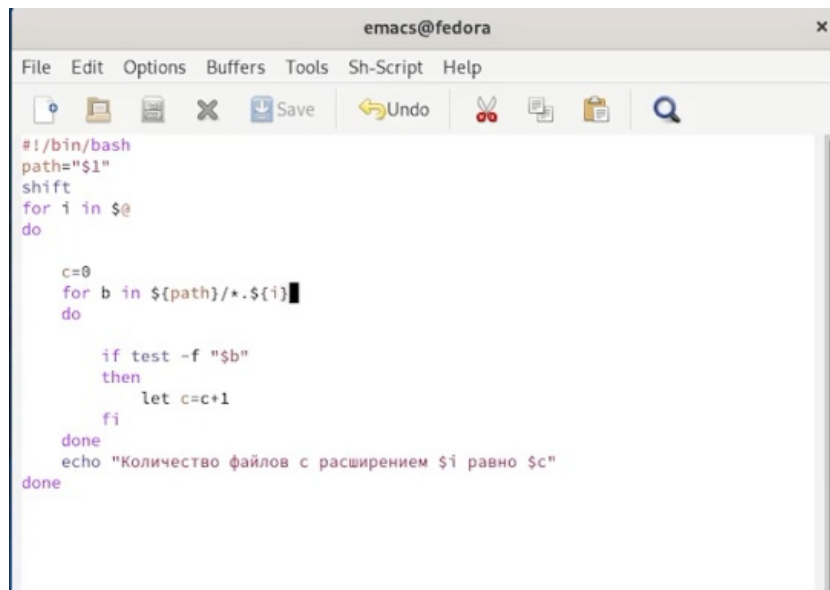


Рис. 3.11: Вводим команды в файл

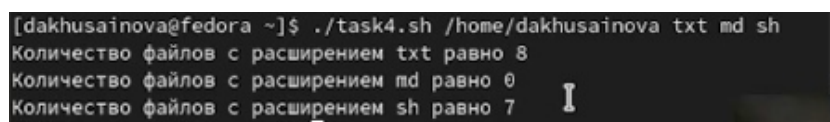


Рис. 3.12: Запускаем

## 4 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`);

## 5 Вывод

Изучили основы программирования в оболочке ОС UNIX/Linux и научились писать небольшие командные файлы.