

Лабораторная работа №10

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

Латыпова Диана. НФИбд-02-21

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Контрольные вопросы	15
5	Выводы	20

Список иллюстраций

3.1	Создание директории	7
3.2	Запуск редактора emacs(1)	7
3.3	Скрипт 1 задания	8
3.4	10_1	9
3.5	Запуск редактора emacs(2)	9
3.6	Скрипт 2 задания	10
3.7	10_2	11
3.8	Запуск редактора emacs(3)	11
3.9	Скрипт 3 задания	12
3.10	10_3	13
3.11	Запуск редактора emacs(4)	13
3.12	Скрипт 4 задания	14
3.13	10_4	14

Список таблиц

1 Цель работы

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

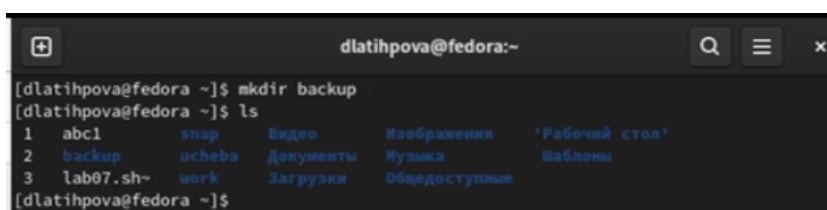
2 Задание

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

1. Я написала скрипт, который при запуске делает резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в домашнем каталоге. При этом файл архивировался архиватором zip.

Для начала я создала директорию backup с помощью команды `mkdir`(рис. 3.1):
mkdir backup



```
dlatihpova@fedora:~  
[dlatihpova@fedora ~]$ mkdir backup  
[dlatihpova@fedora ~]$ ls  
1  abc1      snap      Видео      Изображения  'Рабочий стол'  
2  backup    ucheba    Документы  Музыка       Шаблоны  
3  lab07.sh~ work      Загрузки   Общедоступные  
[dlatihpova@fedora ~]$
```

Рис. 3.1: Создание директории

Затем запустила редактор emacs(рис. 3.2):

emacs 10_1



```
[dlatihpova@fedora ~]$ emacs 10_1
```

Рис. 3.2: Запуск редактора emacs(1)

Написала там скрипт 1 задания(рис. 3.3):

1 zip 10_1.zip 10_1

2 mv 10_1.zip /home/dlatihpova/backup

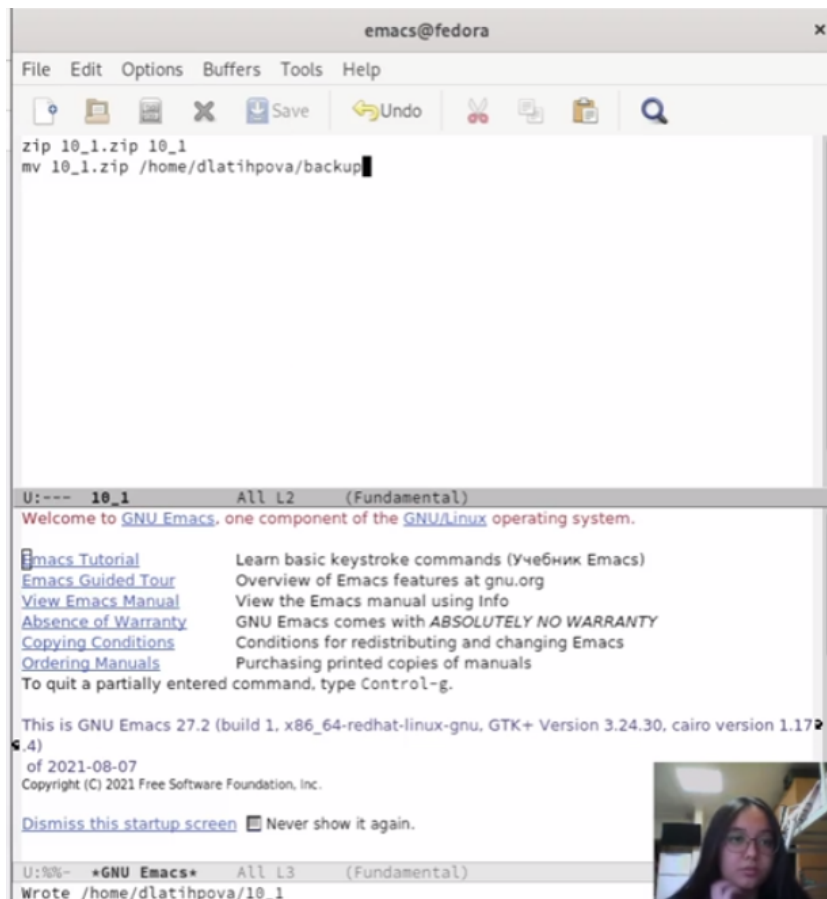


Рис. 3.3: Скрипт 1 задания

После чего закрыла редактор. Изменила код защиты командного файла 10_1, обеспечив доступ к этому файлу по выполнению и проверила работу нашего скрипта(рис. 3.4):

- 1 **chmod +x 10_1**
- 2 **./10_1**
- 3 **cd backup**
- 4 **ls**


```
[dlatihpova@fedora ~]$ chmod +x 10_1
[dlatihpova@fedora ~]$ ./10_1
  adding: 10_1 (deflated 23%)
[dlatihpova@fedora ~]$ cd backup
[dlatihpova@fedora backup]$ ls
10_1.zip
```

Рис. 3.4: 10_1

Все работает. Перешла к выполнению второго задания.

2. Далее я написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять.

Снова запустила редактор emacs(рис. 3.5):

emacs 10_2

```
[dlatihpova@fedora backup]$ cd
[dlatihpova@fedora ~]$ emacs 10_2
```

Рис. 3.5: Запуск редактора emacs(2)

Написала скрипт 2 задания(рис. 3.6):

1 for numb in \$*

2 do echo \$numb

3 done

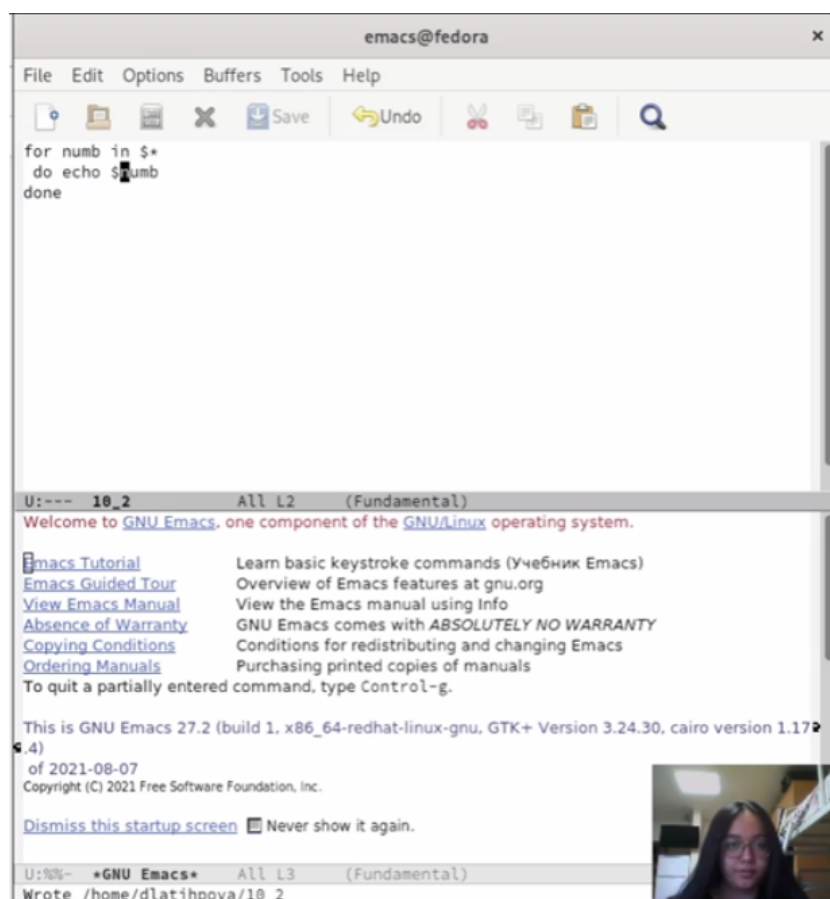


Рис. 3.6: Скрипт 2 задания

Снова предоставила права на выполнение и проверила работу скрипта(рис. 3.7):

1 **chmod +x 10_2**

2 **./10_2 1 2 3 4 5 6 7 8 9 10 11**

```
[dlatihpova@fedora ~]$ chmod +x 10_2
[dlatihpova@fedora ~]$ ./10_2 1 2 3 4 5 6 7 8 9 10 11
1
2
3
4
5
6
7
8
9
10
11
[dlatihpova@fedora ~]$
```

Рис. 3.7: 10_2

Скрипт работает, выводит последовательно наши аргументы. Переходим к выполнению 3 задания.

3. Написала командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). В результате он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

Запустила редактор `emacs`(рис. 3.8):

emacs 10_3

```
[dlatihpova@fedora ~]$ emacs 10_3
```

Рис. 3.8: Запуск редактора `emacs`(3)

Написала скрипт 3 задания на примере скрипта в теоретической части лабораторной работы(рис. 3.9):

```
1 for A in *
2 do if test -d $A
3 then echo $A: is a directory
4 else echo -n $A: "is a file and"
```

```

5 if test -w $A
6 then echo writeable
7 elif test -r $A
8 then echo readable
9 else echo neither readable nor writeable
10 fi
11 fi
12 done

```

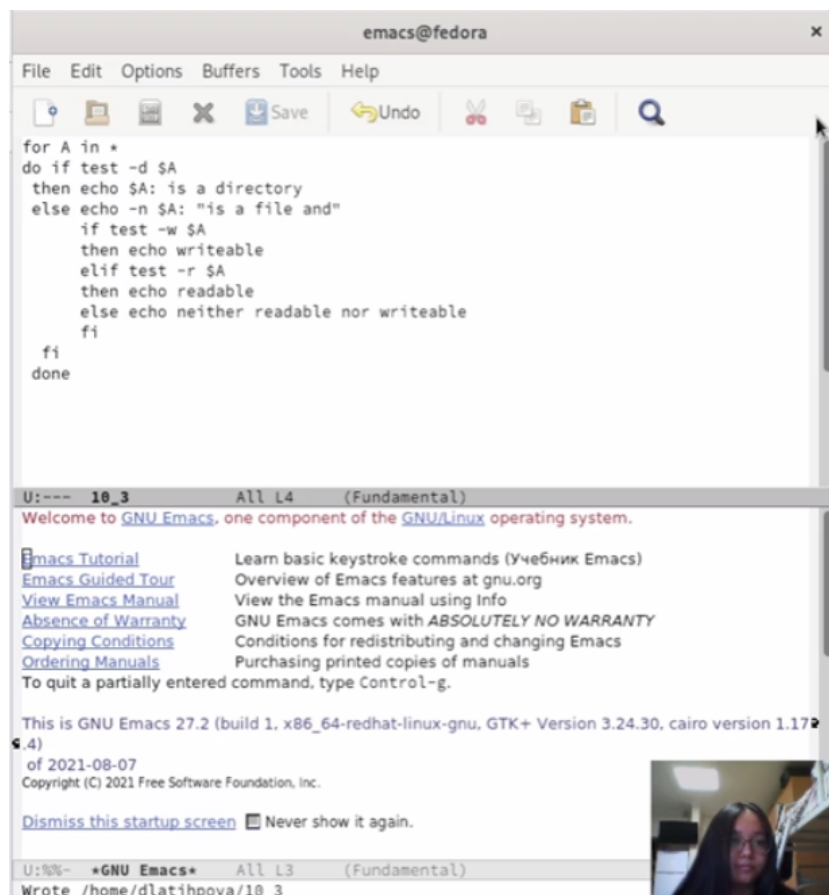


Рис. 3.9: Скрипт 3 задания

Снова предоставила права на выполнение и проверила работу скрипта(рис. 3.10):

```

1 chmod +x 10_3
2 ./10_3

```

```
[dlatihpova@fedora ~]$ chmod +x 10_3
[dlatihpova@fedora ~]$ ./10_3
1: is a file andwriteable
10_1: is a file andwriteable
10_2: is a file andwriteable
10_3: is a file andwriteable
2: is a file andwriteable
3: is a file andwriteable
abcl: is a file andwriteable
backup: is a directory
lab07.sh~: is a file andwriteable
snap: is a directory
ucheba: is a directory
work: is a directory
Видео: is a directory
Документы: is a directory
Загрузки: is a directory
Изображения: is a directory
Музыка: is a directory
Общедоступные: is a directory
./10_3: строка 2: test: Рабочий: ожидается бинарный оператор
Рабочий стол: is a file and./10_3: строка 5: test: Рабочий: ожидае
атор
./10_3: строка 7: test: Рабочий: ожидается бинарный оператор
neither readable nor writeable
Шаблоны: is a directory
[dlatihpova@fedora ~]$
```

Рис. 3.10: 10_3

И наконец, приступила к выполнению 4 задания.

4. Я написала 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.

Запустила редактор emacs(рис. 3.11):

emacs 10_4

```
[dlatihpova@fedora ~]$ emacs 10_4
```

Рис. 3.11: Запуск редактора emacs(4)

Написала скрипт 4 задания(рис. 3.12):

- 1 **echo "Directory path:"**
- 2 **read dir**
- 3 **echo "File format:"**
- 4 **read format**

4 Контрольные вопросы

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

Оболочка операционной системы (от англ. shell — оболочка) — интерпретатор команд операционной системы (ОС), обеспечивающий интерфейс для взаимодействия пользователя с функциями системы.

- Bash — самая распространённая оболочка под Linux. Она ведёт историю команд и предоставляет возможность их редактирования;
- pdksh — клон korn shell, хорошо известной оболочки в системах UNIX;
- tcsh — улучшенная версия >C shell;
- zsh — новейшая из перечисленных здесь оболочек; реализует улучшенное дополнение и другие удобные функции.

У каждой оболочки свой синтаксис.

2. Что такое POSIX?

POSIX (произносится как «позикс») — это интерфейс портативных операционных систем. Во-первых, нужно обозначить область действия понятия «портативность», в этом конкретном случае, и определиться с понятием «интерфейс». Чтобы выяснить это, необходимо отталкиваться от того, что оба понятия неразрывно связаны.

3. Как определяются переменные и массивы в языке программирования bash?

Bash-массивы имеют только численные индексы, но они не обязательны к использованию, то есть вы не должны определять значения всех индексов в явном виде. Массив целиком может быть определен путем заключения записей в круглые скобки.

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.

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.

Необходимо обеспечить возможность, однажды написав программу, многократно ее использовать, вводя каждый раз другие данные. Такая гибкость в языке обеспечивается оператором Read. Этим оператором вводится информация с клавиатуры.

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

let это встроенная функция bash, которая позволяет производить базовые арифметические операции.

Используется следующим образом:

let

+, -, *, / -Сложение, вычитание, умножение, деление

var++ -Увеличение переменной на 1

var-- -Уменьшение переменной на 1

% -Модуль: возвращает остаток от деления

6. Что означает операция (())?

Условия оболочки `bash`, в двойные скобки —(()).

7. Какие стандартные имена переменных Вам известны?

Общие имена временных переменных: `i`, `j`, `k`, `m` и `n` для целых чисел; `c`, `d` и `e` для символов.

8. Что такое метасимволы?

Метасимволы – это специальные символы, имеющие особое значение, такие как подстановочный символ, символ повторения, символ несовпадения или диапазон символов `' < > * ? | " &`

9. Как экранировать метасимволы?

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 -lrf` Если есть `d`, то является файл каталогом

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

Команда `SET` используется для просмотра и изменения переменных среды окружения в командной строке Windows. Переменные окружения - это переменные, принимаемые значения которых характеризуют среду, в которой выполняется текущая программа - пути системных файлов, сведения об аппаратных средствах, каталоги пользователя и т.п. Значения переменных среды формируются в процессе загрузки Windows, регистрации пользователя в системе, при выполнении отдельных процессов или с помощью команды `SET`.

Команды `declare` и `typeset` являются встроенными и предназначены для наложения ограничений на переменные. Это попытка контроля над типами, которая

имеется во многих языках программирования. Команды абсолютно идентичны друг другу и являются синонимами.

Команда `unset` удаляет переменную, фактически – устанавливает ее значение в `null`. Обратите внимание: эта команда не может сбрасывать позиционные параметры (аргументы).

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.

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

\$1-\$9 Это параметры командной строки. Т.е. то, что пользователь ввел через пробел после названия самого скрипта.

\$0 В этой переменной лежит путь и имя скрипта, который запустил пользователь.

\$# Количество параметров, переданных скрипту из командной строки.

\$? Код возврата (exit code, result code), с которым завершилась предыдущая команда. Как правило, код возврата “0” означает удачное выполнение команды, а все, что отлично от нуля – различные ошибки, причем коды ошибок четко специфичны для выполняемой команды.

\$\$ PID – код процесса, в котором выполняется данный скрипт.

#! PID последнего процесса, который был запущен в фоне. Например, если Вы запускаете в фоне какой-то процесс, то Вам, возможно, хотелось-бы знать его PID.

\$* Все параметры, переданные из командной строки, в одну строку.

5 Выводы

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