

Лабораторная работа №8

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

Мальсагов Акрамат Абу-Бакарович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	10
4	Контрольные вопросы	11

Список иллюстраций

2.1	Код 1 скрипта	6
2.2	Результат работы 1 скрипта	7
2.3	Код 2 скрипта	7
2.4	Работа скрипта	8
2.5	Код 3 скрипта	8
2.6	Результат работы скрипта	8
2.7	Код 4 скрипта	9
2.8	Работа скрипта	9

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Создал script1 и открыл его в emacs. Написал программу, которая при запуске будет делать резервную копию самого себя. (рис. 2.1)

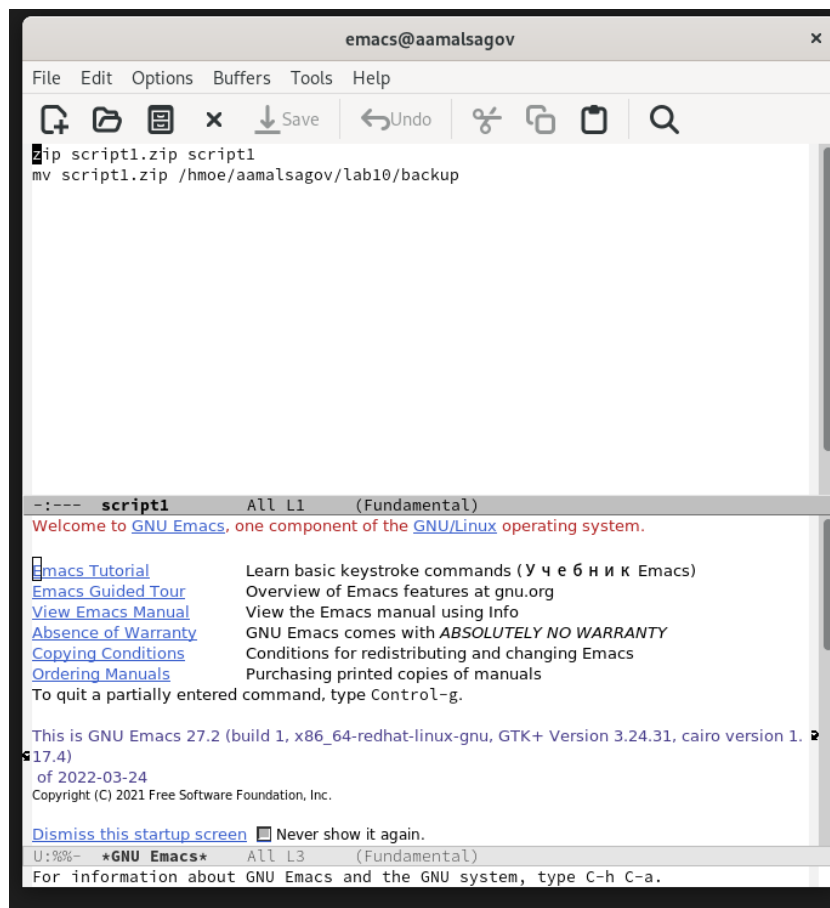


Рис. 2.1: Код 1 скрипта

2. Проверил его работу. (рис. 2.2)

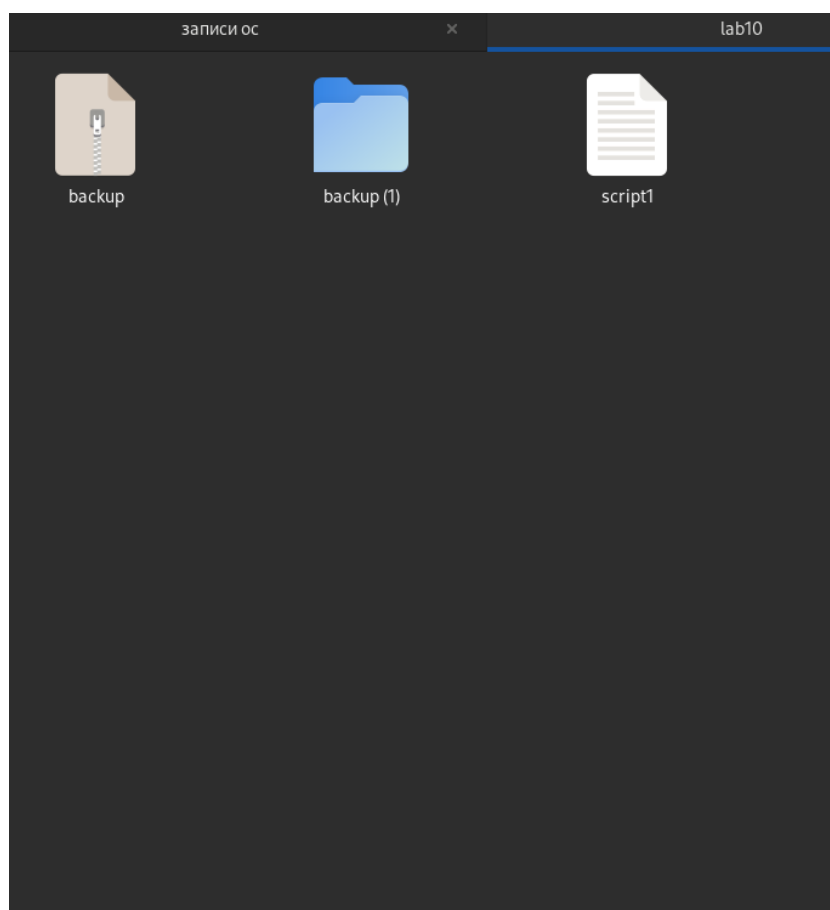


Рис. 2.2: Результат работы 1 скрипта

3. Создал script2 и открыл его в emacs. Написал программу, обрабатывающая любое произвольное число аргументов командной строки(рис. 2.3)

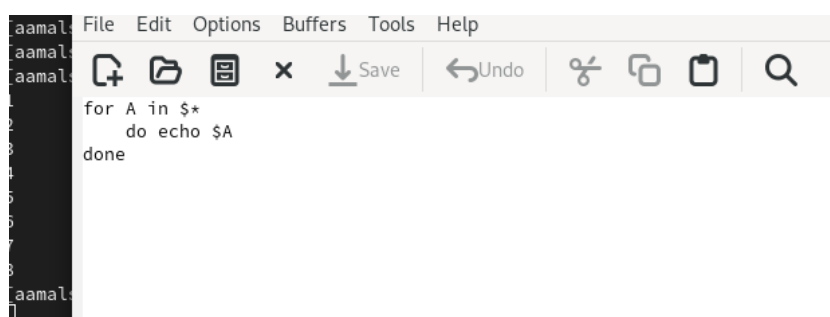


Рис. 2.3: Код 2 скрипта

4. Запустил скрипт.(рис. 2.4)

```

aamalsagov@aamalsagov lab10]$ emacs script2
aamalsagov@aamalsagov lab10]$ chmod +x script2
aamalsagov@aamalsagov lab10]$ ./script2 1 2 3 4 5 6 7 8
1
2
3
4
5
6
7
8
aamalsagov@aamalsagov lab10]$

```

Рис. 2.4: Работа скрипта

5. Создал script3 и открыл его в emacs. Написал программу, аналог команды ls.(рис. 2.5)

```

backup (1):
./script3: c for A in *
./script3: c do if test -d $A
script1: is then echo $A: is a directory
script1~: is else echo -n $A: "is a file and "
script2: is if test -w $A
script3: is then echo writeable
script3~: is elif test -r $A
aamalsagov then echo eadable
else echp neither readable nor writeable
fi
done

```

Рис. 2.5: Код 3 скрипта

6. Проверил работу скрипта. (рис. 2.6)

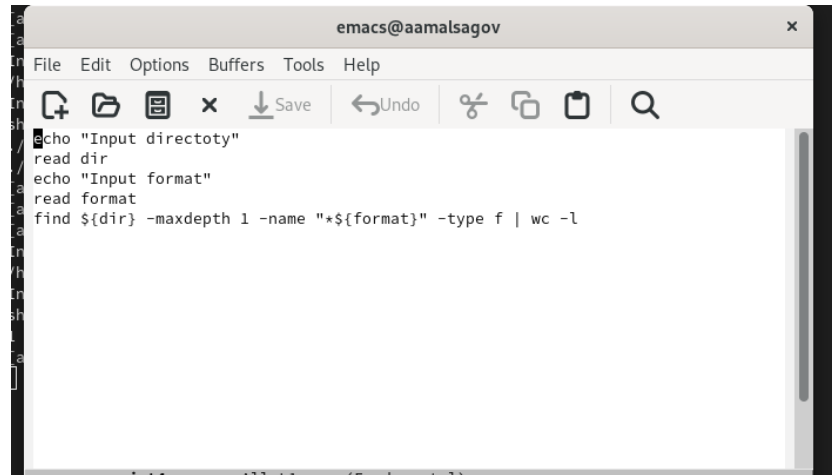
```

aamalsagov@aamalsagov lab10]$ chmod +x script3
aamalsagov@aamalsagov lab10]$ ./script3
./script3: строка 1: синтаксическая ошибка рядом с неожиданным маркером «*»
./script3: строка 1: `for Ain *'
aamalsagov@aamalsagov lab10]$ ./script3
./script3: строка 1: синтаксическая ошибка рядом с неожиданным маркером «*»
./script3: строка 1: `for Ain *'
aamalsagov@aamalsagov lab10]$ emacs script3
aamalsagov@aamalsagov lab10]$ ./script3
backup: is a file and writeable
./script3: строка 2: test: backup: ожидается бинарный оператор
backup (1): is a file and ./script3: строка 5: test: backup: ожидается бинарный оператор
./script3: строка 7: test: backup: ожидается бинарный оператор
./script3: строка 9: echp: команда не найдена
script1: is a file and writeable
script1~: is a file and writeable
script2: is a file and writeable
script3: is a file and writeable
script3~: is a file and writeable
aamalsagov@aamalsagov lab10]$ emacs script3
aamalsagov@aamalsagov lab10]$

```

Рис. 2.6: Результат работы скрипта

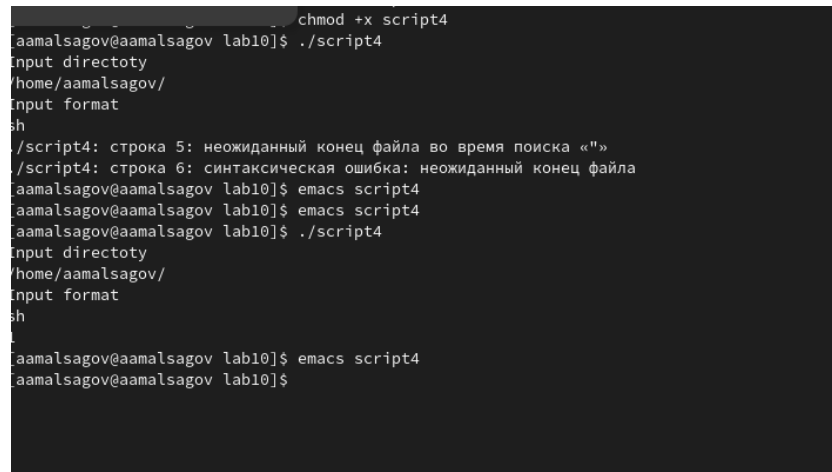
7. Создал script4 и открыл его в emacs. Написал программу, которая просит ввести путь к директории и формат файла, а затем выдает количество файлов с данным форматом в данной директории.(рис. 2.7)

The image shows a screenshot of the Emacs editor window titled 'emacs@aamalsagov'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations and editing. The main text area contains the following shell script code:

```
echo "Input directoty"
read dir
echo "Input format"
read format
find ${dir} -maxdepth 1 -name "*${format}" -type f | wc -l
```

Рис. 2.7: Код 4 скрипта

8. Запустил скрипт.(рис. 2.8)

The image shows a screenshot of a terminal window. The user has executed the following commands:

```
chmod +x script4
./script4
```

The output of the script is:

```
Input directoty
/home/aamalsagov/
Input format
sh
./script4: строка 5: неожиданный конец файла во время поиска «"»
./script4: строка 6: синтаксическая ошибка: неожиданный конец файла
```

The user then runs 'emacs script4' and the script again, which produces the same error messages. Finally, the user runs 'emacs script4' again, and the prompt returns to the shell without further output.

Рис. 2.8: Работа скрипта

3 Выводы

Мы научились писать небольшие команды файлы.

4 Контрольные вопросы

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 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. POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

3. `mark=/usr/andy/bin`

Данная команда присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка-символов.

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"
```

4. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Команда `read` позволяет читать значения переменных со стандартного ввода
5. Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки — (()).
7. Переменные `PS1` и `PS2` 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 `>`. 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, 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. Такие символы, как ' < > * ? | " &, являются метасимволами и имеют для командного процессора специальный смысл
 9. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом.
 10. Командный файл можно создать с помощью какого-либо редактора, затем сделать его исполняемым и запустить его из терминала, введя “./название файла”.
 11. помощью ключевого слова *function*.
 12. Вводим команду *ls -lrt* и если первым в правах доступа стоит *d* то это каталог. Иначе это файл.
 13. Для создания массива используется команда *set* с флагом *-A*. Если использовать *typeset -i* для объявления и присвоения переменной, то при последующем её применении она станет целой. Изъять переменную из программы можно с помощью команды *unset*.
 14. 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.

15.

- `$*` — отображается вся командная строка или параметры оболочки;
- `$?` — 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`.