

Лабораторная работа №10

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

Перевощиков Данил Алексеевич

Содержание

| | | |
|----------|----------------------------|-----------|
| 1 | Цель работы | 5 |
| 2 | Ход работы | 6 |
| 3 | Вывод | 10 |
| 4 | Контрольные вопросы | 11 |

Список иллюстраций

| | | |
|-----|--|---|
| 2.1 | Первый скрипт. | 6 |
| 2.2 | Тестирование первого скрипта. | 6 |
| 2.3 | Второй скрипт. | 7 |
| 2.4 | Тестирование второго скрипта. | 7 |
| 2.5 | Третий скрипт. | 8 |
| 2.6 | Тестирование третьего скрипта. | 8 |
| 2.7 | Четвертый скрипт. | 9 |
| 2.8 | Тестирование четвертого скрипта. | 9 |

Список таблиц

1 Цель работы

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

2 Ход работы

1. Написали 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 архивируется архиватором zip.(рис. 2.1)

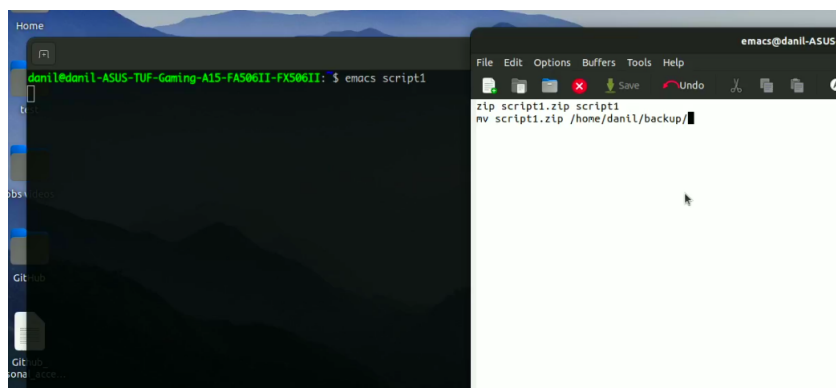


Рис. 2.1: Первый скрипт.

Далее делаем файл script1 исполняемым и тестируем его.(рис. 2.2)

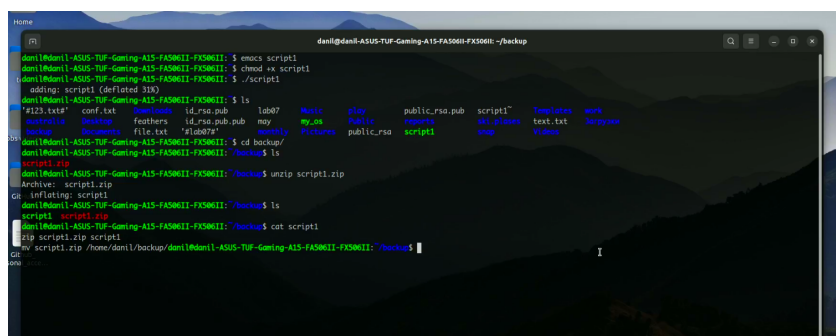


Рис. 2.2: Тестирование первого скрипта.

2. Написали пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Наш скрипт последовательно печатает значения всех переданных аргументов.(рис. 2.3)

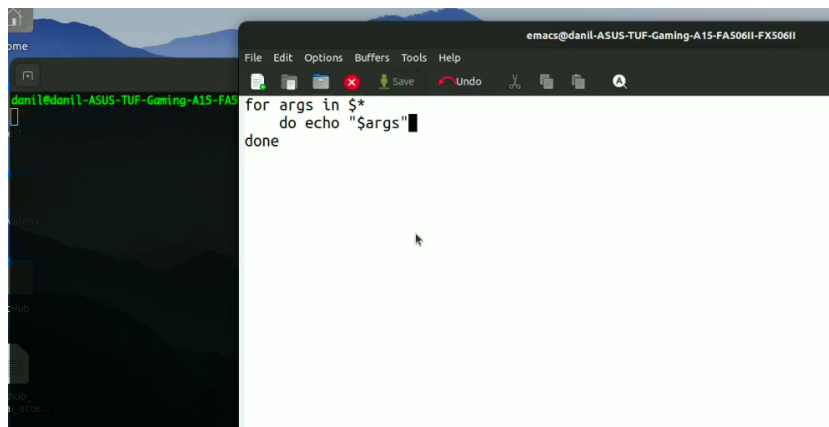


Рис. 2.3: Второй скрипт.

Далее делаем файл script2 исполняемым и тестируем его.(рис. 2.4)

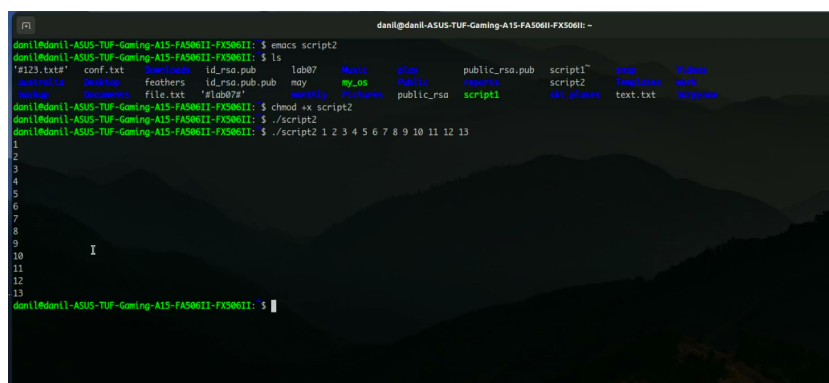
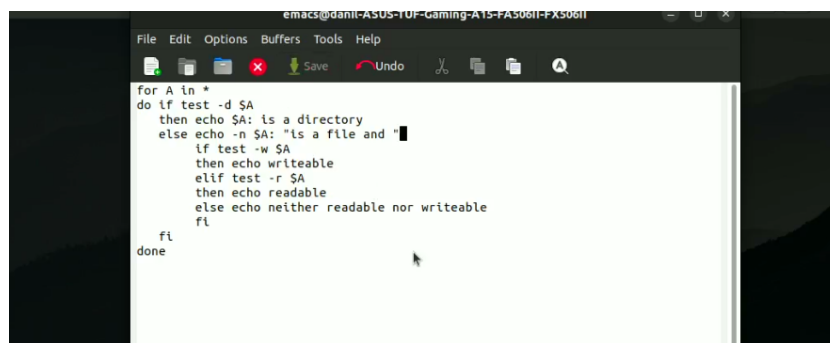


Рис. 2.4: Тестирование второго скрипта.

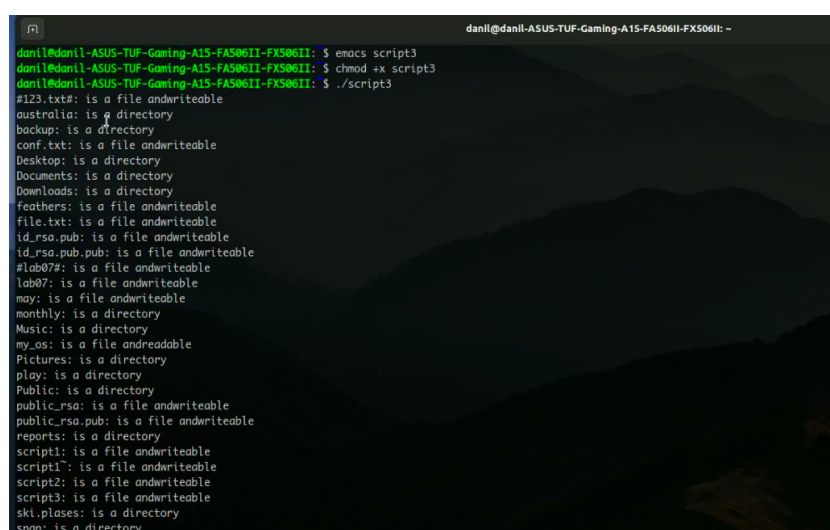
3. Написали командный файл — аналог команды ls (без использования самой этой команды и команды dir). Он выдает информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога.(рис. 2.5)



```
for A in *
do if test -d $A
then echo $A: is a directory
else echo -n $A: "is a file and "
if test -w $A
then echo writeable
elif test -r $A
then echo readable
else echo neither readable nor writeable
fi
fi
done
```

Рис. 2.5: Третий скрипт.

Далее делаем файл script3 исполняемым и тестируем его.(рис. 2.6)



```
danil@danil-ASUS-TUF-Gaming-A15-FA506II-FX506II: ~$ emacs script3
danil@danil-ASUS-TUF-Gaming-A15-FA506II-FX506II: ~$ chmod +x script3
danil@danil-ASUS-TUF-Gaming-A15-FA506II-FX506II: ~$ ./script3
#123.txt: is a file andwriteable
australia: is a directory
backup: is a directory
conf.txt: is a file andwriteable
Desktop: is a directory
Documents: is a directory
Downloads: is a directory
feathers: is a file andwriteable
file.txt: is a file andwriteable
id_rsa.pub: is a file andwriteable
id_rsa.pub.pub: is a file andwriteable
#lab07#: is a file andwriteable
lab07: is a file andwriteable
may: is a file andwriteable
monthly: is a directory
Music: is a directory
my_os: is a file andreadable
Pictures: is a directory
play: is a directory
Public: is a directory
public_rsa: is a file andwriteable
public_rsa.pub: is a file andwriteable
reports: is a directory
script1: is a file andwriteable
script1: is a file andwriteable
script2: is a file andwriteable
script3: is a file andwriteable
ski.places: is a directory
snap: is a directory
```

Рис. 2.6: Тестирование третьего скрипта.

4. Написали 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. 2.7)

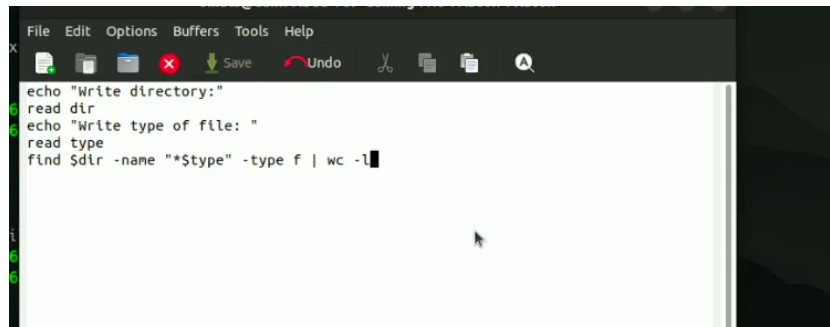


Рис. 2.7: Четвертый скрипт.

Далее делаем файл script3 исполняемым и тестируем его. Для этого в директории backup создаем несколько файлов формата .txt и затем проверяем их кол-во скриптом.(рис. 2.8)

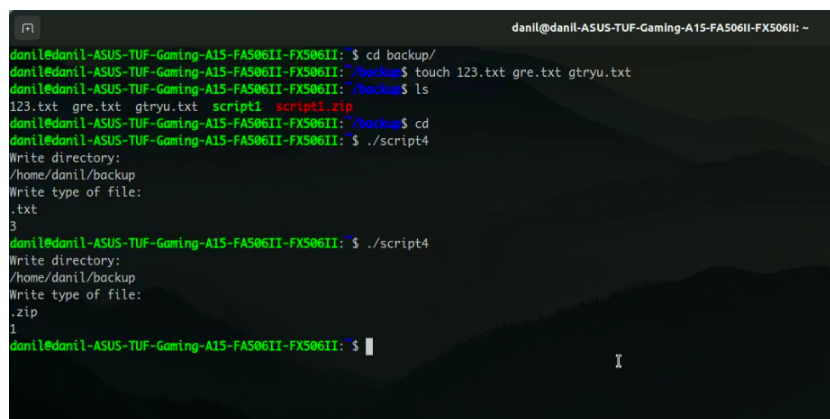


Рис. 2.8: Тестирование четвертого скрипта.

3 Вывод

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

4 Контрольные вопросы

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

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. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол \$.

4. Каково назначение операторов let и read?

Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению.

Команда read позволяет считывать значения, введенные с клавиатуры.

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.

8. Что такое метасимволы?

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?

Сначала нужно воспользоваться любым редактором и записать в файл какой-либо код, затем сделать этот файл исполняемым командой `chmod +x`. После чего запустить его командой `./`.

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.

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

Задать условие:

```
if <имя файла> -d
```

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

Значение всех переменных можно просмотреть с помощью команды `set`.

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 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.

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.

- $\$*$ — 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`.