

# **Лабораторная работа №10**

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

Медникова Екатерина Михайловна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	9
4	Контрольные вопросы	10
	Список литературы	17

## Список иллюстраций

2.1	Скрипт . . . . .	6
2.2	Результат . . . . .	6
2.3	Командный файл . . . . .	6
2.4	Результат . . . . .	7
2.5	Командный файл . . . . .	7
2.6	Результат . . . . .	7
2.7	Результат . . . . .	8
2.8	Командный файл . . . . .	8
2.9	Результат . . . . .	8

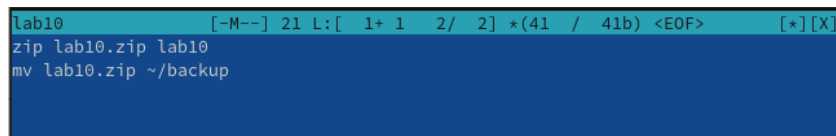
## Список таблиц

# 1 Цель работы

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

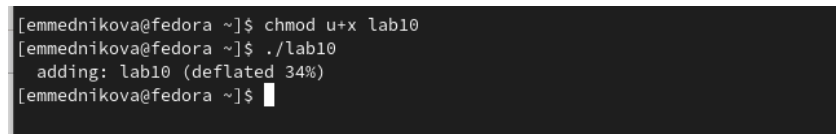
## 2 Выполнение лабораторной работы

1. Написала скрипт, который при запуске делает резервную копию самого себя.



```
lab10 [-M--] 21 L: [ 1+ 1 2/ 2] *(41 / 41b) <EOF> [*][X]  
zip lab10.zip lab10  
mv lab10.zip ~/backup
```

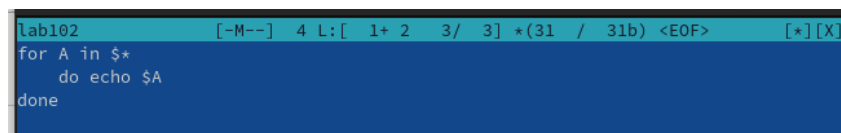
Рис. 2.1: Скрипт



```
[emmednikova@fedora ~]$ chmod u+x lab10  
[emmednikova@fedora ~]$ ./lab10  
adding: lab10 (deflated 34%)  
[emmednikova@fedora ~]$
```

Рис. 2.2: Результат

2. Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки.



```
lab102 [-M--] 4 L: [ 1+ 2 3/ 3] *(31 / 31b) <EOF> [*][X]  
for A in $*  
do echo $A  
done
```

Рис. 2.3: Командный файл

```
[emmednikova@fedora ~]$ chmod u+x lab102
[emmednikova@fedora ~]$ ./lab102 1 2 3 4 5 6 7 8 9 10 11
1
2
3
4
5
6
7
8
9
10
11
[emmednikova@fedora ~]$
```

Рис. 2.4: Результат

3. Написала командный файл - аналог команды ls.

```
lab103 [-M--] 4 L: [ 1+11 12/ 12] *(262 / 262b) <EOF> [*][X]
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: Командный файл

```
[emmednikova@fedora ~]$ chmod u+x lab103
[emmednikova@fedora ~]$ ./lab103
abcl: is a file andwriteable
backup: is a file andwriteable
bin: is a directory
conf.txt: is a file andwriteable
feathers: is a file andwriteable
#filel#: is a file andwriteable
file.txt: is a file andwriteable
lab07.sh: is a file andwriteable
lab07.sh~: is a file andwriteable
lab10: is a file andwriteable
lab102: is a file andwriteable
lab103: is a file andwriteable
may: is a file andwriteable
monthly: is a directory
play: is a directory
report.md: is a file andreadable
reports: is a directory
ski.plases: is a directory
```

Рис. 2.6: Результат

```

may: is a file andwriteable
monthly: is a directory
play: is a directory
report.md: is a file andreadable
reports: is a directory
ski.places: is a directory
test1: is a directory
test1.txt: is a file andwriteable
test2: is a directory
text.txt: is a file andwriteable
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
./lab103: строка 2: test: Рабочий: ожидается бинарный оператор
Рабочий стол: is a file and./lab103: строка 5: test: Рабочий: ожидается бинарный оператор
./lab103: строка 7: test: Рабочий: ожидается бинарный оператор
neither readable nor writeable
Шаблоны: is a directory

```

Рис. 2.7: Результат

4. Написала командный файл, который получает в качестве аргумента командной строки формат файла.

```

lab104 [----] 54 L:[ 1+ 4 5/ 5] *(118 / 118b) <E0F> [*][X]
echo "Input directory"
read dir
echo "Input format"
read format
find $dir -maxdepth 1 -name "*${type}" -type f | wc -l

```

Рис. 2.8: Командный файл

```

[emmednikova@fedora ~]$ chmod u+x lab104
[emmednikova@fedora ~]$ ./lab104
Input directory
work
Input format
.doc
0
[emmednikova@fedora ~]$ ./lab104
Input directory
work
Input format
.txt
0

```

Рис. 2.9: Результат



## **3 Выводы**

Изучила основы программирования в оболочке ОС UNIX/Linux. Научилась 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 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. *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 \$. 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.

### *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 `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара. Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Команда `let` не ограничена простыми арифметическими выражениями. Команда `read` позволяет читать значения переменных со стандартного ввода.

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

$!(\text{exp})$  - Если  $\text{exp}$  равно 0, то возвращает 1; иначе 0  
 $!(\text{exp1} != \text{exp2})$  - Если  $\text{exp1}$  не равно  $\text{exp2}$ , то возвращает 1; иначе 0  
 $\% (\text{exp1} \% \text{exp2})$  - Возвращает остаток от деления  $\text{exp1}$  на  $\text{exp2}$   
 $\% = (\text{var} \% \text{exp})$  - Присваивает остаток от деления  $\text{var}$  на  $\text{exp}$  переменной  $\text{var}$   
 $\& (\text{exp1} \& \text{exp2})$  - Возвращает побитовое AND выражений  $\text{exp1}$  и  $\text{exp2}$   
 $\&\& (\text{exp1} \&\& \text{exp2})$  - Если и  $\text{exp1}$  и  $\text{exp2}$  не равны нулю, то возвращает 1; иначе 0  
 $\& = (\text{var} \& = \text{exp})$  - Присваивает переменной  $\text{var}$  побитовое AND  $\text{var}$  и  $\text{exp}$   
 $(\text{exp1} * \text{exp2})$  - Умножает  $\text{exp1}$  на  $\text{exp2}$   
 $= (\text{var} = \text{exp})$  - Умножает  $\text{exp}$  на значение переменной  $\text{var}$  и присваивает результат переменной  $\text{var}$   
 $(\text{exp1} + \text{exp2})$  - Складывает  $\text{exp1}$  и  $\text{exp2}$   
 $+= (\text{var} += \text{exp})$  - Складывает  $\text{exp}$  со значением переменной  $\text{var}$  и результат присваивает переменной  $\text{var}$   
 $(-\text{exp})$  - Операция отрицания  $\text{exp}$  (унарный минус)  
 $(\text{exp1} - \text{exp2})$  - Вычитает  $\text{exp2}$  из  $\text{exp1}$   
 $-= (\text{var} -= \text{exp})$  - Вычитает  $\text{exp}$  из значения переменной  $\text{var}$  и присваивает результат переменной  $\text{var}$   
 $/ (\text{exp} / \text{exp2})$  - Делит  $\text{exp1}$  на  $\text{exp2}$   
 $/= (\text{var} /= \text{exp})$  - Делит значение переменной  $\text{var}$  на  $\text{exp}$  и присваивает результат переменной  $\text{var}$   
 $< (\text{exp1} < \text{exp2})$  - Если  $\text{exp1}$  меньше, чем  $\text{exp2}$ , то возвращает 1, иначе возвращает 0  
 $\ll (\text{exp1} \ll \text{exp2})$  - Сдвигает  $\text{exp1}$  влево на  $\text{exp2}$  бит  
 $\ll = (\text{var} \ll = \text{exp})$  - Побитовый сдвиг влево значения переменной  $\text{var}$  на  $\text{exp}$   
 $\leq (\text{exp1} \leq \text{exp2})$  - Если  $\text{exp1}$  меньше или равно  $\text{exp2}$ , то возвращает 1; иначе возвращает 0  
 $= (\text{var} = \text{exp})$  - Присваивает значение  $\text{exp}$  переменной  $\text{var}$   
 $== (\text{exp1} == \text{exp2})$  - Если  $\text{exp1}$  равно  $\text{exp2}$ , то возвращает 1; иначе возвращает 0  
 $(\text{exp1} > \text{exp2})$  - 1, если  $\text{exp1}$  больше, чем  $\text{exp2}$ ; иначе 0

(exp1 >= exp2) - 1, если exp1 больше или равно exp2; иначе 0

(exp » exp2) - Сдвигает exp1 вправо на exp2 бит

(var »=exp) - Побитовый сдвиг вправо значения переменной var на exp

^ (exp1 ^ exp2) - Исключающее OR выражений exp1 и exp2

^= (var ^= exp) - Присваивает переменной var побитовое XOR var и exp

(exp1 | exp2) - Побитовое OR выражений exp1 и exp2

|= (var |= exp) - Присваивает переменной var результат операции XOR var и exp

**|| (exp1 || exp2) - 1, если или exp1 или exp2 являются ненулевыми значениями; иначе 0**

(~exp) - Побитовое дополнение до exp

#### 6. Что означает операция (( ))?

Подобно C оболочка bash может присваивать переменной любое значение, а произвольное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для облегчения программирования можно записывать условия оболочки bash в двойные скобки — (( )).

#### 7. Какие стандартные имена переменных Вам известны?

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. Что такое метасимволы?

Такие символы, как ' < > \* ? | " &, являются метасимволами и имеют для командного процессора специальный смысл.

#### 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 \$, ', ,, " .

#### 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.

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

Нужно использовать команду `ls -lrt`. Если в правах доступа первой стоит буква `d`, то это каталог, иначе - файл.

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

Команда `set` позволяет просмотреть значение всех переменных. Команда `unset` с флагом `-f` удаляет функцию.

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 использовании в командном файле комбинации символов `$#` вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.

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`.



## **Список литературы**