

# **Лабораторная работа №10**

**Дисциплина: Операционные системы**

Кондратьев Арсений Вячеславович

23.09.2022

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>4</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
<b>4</b>	<b>Выводы</b>	<b>14</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>15</b>

# 1 Цель работы

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

## 2 Теоретическое введение

`mark` - присваивает значение строки символов

`let` - является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению

`break` - прерывание циклов

### 3 Выполнение лабораторной работы

1. Написал скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в домашнем каталоге. Для этого использовал bzip2(рис.3.1)(рис.3.2)

Создал файл для скрипта backup.sh и дал права на исполнение

```
touch backup.sh  
chmod +x *.sh
```

Присвоил переменной название файла скрипта, затем создал каталог, в которую требуется сохранить резервную копию, заархивировал файл скрипта с помощью bzip2 и переместил архив в каталог backup

```
name="bachup.sh"  
mkdir ~/backup  
bzip2 -k ${name}  
mv ${name}.bz2 ~/backup/
```

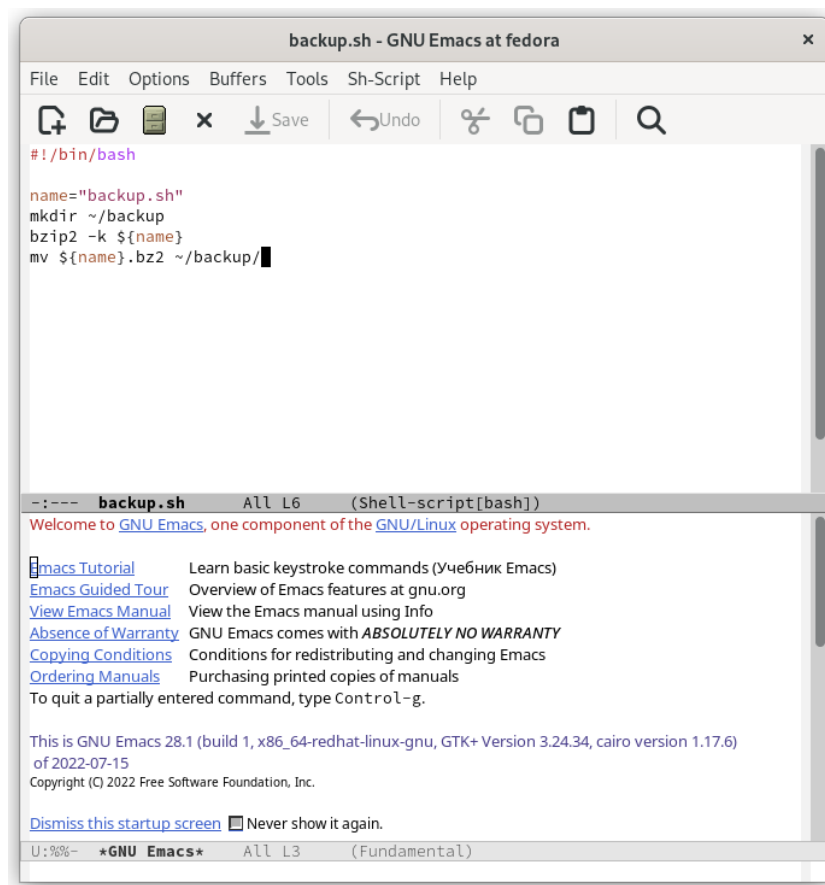
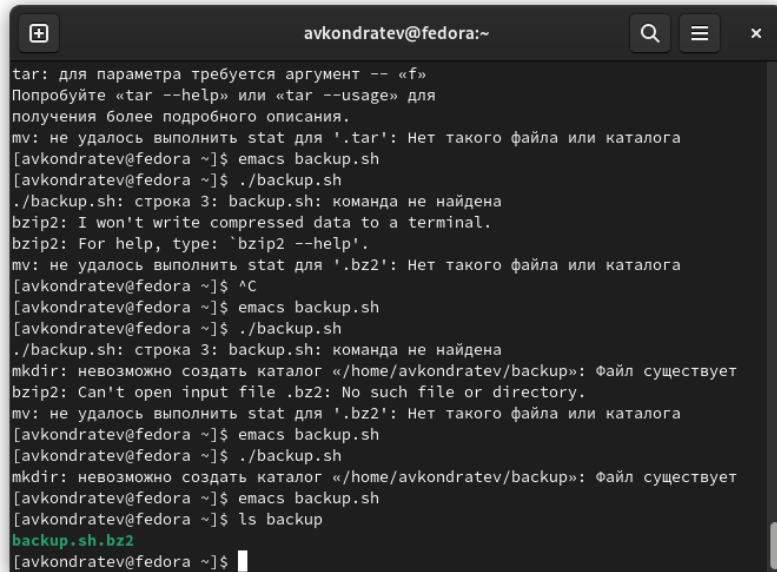


Figure 3.1: Написанный в Emacs скрипт



```
tar: для параметра требуется аргумент -- «f»
Попробуйте «tar --help» или «tar --usage» для
получения более подробного описания.
mv: не удалось выполнить stat для '.tar': Нет такого файла или каталога
[avkondratev@fedora ~]$ emacs backup.sh
[avkondratev@fedora ~]$ ./backup.sh
./backup.sh: строка 3: backup.sh: команда не найдена
bzip2: I won't write compressed data to a terminal.
bzip2: For help, type: `bzip2 --help'.
mv: не удалось выполнить stat для '.bz2': Нет такого файла или каталога
[avkondratev@fedora ~]$ ^C
[avkondratev@fedora ~]$ emacs backup.sh
[avkondratev@fedora ~]$ ./backup.sh
./backup.sh: строка 3: backup.sh: команда не найдена
mkdir: невозможно создать каталог «/home/avkondratev/backup»: Файл существует
bzip2: Can't open input file .bz2: No such file or directory.
mv: не удалось выполнить stat для '.bz2': Нет такого файла или каталога
[avkondratev@fedora ~]$ emacs backup.sh
[avkondratev@fedora ~]$ ./backup.sh
mkdir: невозможно создать каталог «/home/avkondratev/backup»: Файл существует
[avkondratev@fedora ~]$ emacs backup.sh
[avkondratev@fedora ~]$ ls backup
backup.sh.bz2
[avkondratev@fedora ~]$
```

Figure 3.2: Результат: резервная копия в архиве в каталоге

2. Написал скрипт, который последовательно распечатывает значения всех переданных аргументов(рис.3.3)(рис.3.4)

Циклом for прошелся по переданным аргументам и с помощью echo вывел их

```
for i in $@
do echo $i
done
```

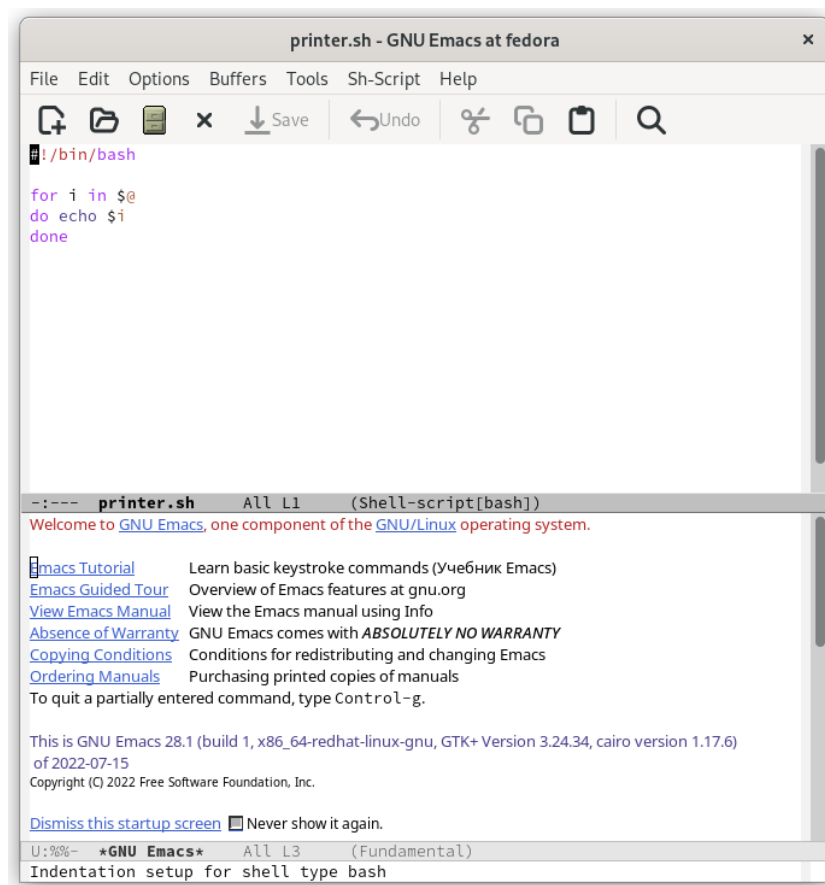
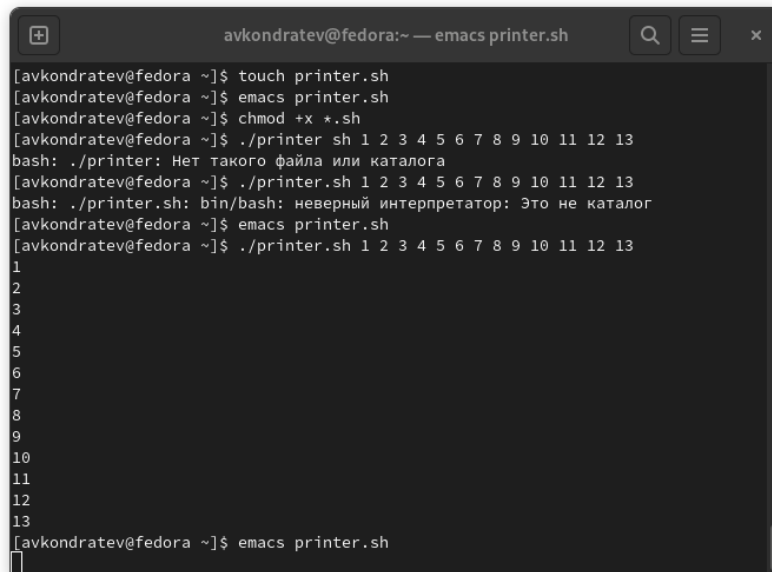


Figure 3.3: Написанный в Emacs скрипт





```
avkondratev@fedora:~ — emacs printer.sh
[avkondratev@fedora ~]$ touch printer.sh
[avkondratev@fedora ~]$ emacs printer.sh
[avkondratev@fedora ~]$ chmod +x *.sh
[avkondratev@fedora ~]$ ./printer.sh 1 2 3 4 5 6 7 8 9 10 11 12 13
bash: ./printer: Нет такого файла или каталога
[avkondratev@fedora ~]$ ./printer.sh 1 2 3 4 5 6 7 8 9 10 11 12 13
bash: ./printer.sh: bin/bash: неверный интерпретатор: Это не каталог
[avkondratev@fedora ~]$ emacs printer.sh
[avkondratev@fedora ~]$ ./printer.sh 1 2 3 4 5 6 7 8 9 10 11 12 13
1
2
3
4
5
6
7
8
9
10
11
12
13
[avkondratev@fedora ~]$ emacs printer.sh
```

Figure 3.4: Результат: переданные аргументы выводятся на экран

3. Написал аналог команды ls, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога(рис.3.5)(рис.3.6)

Присвоил переменной path путь до каталога, циклом for прошелся по всем файлам в этом каталоге и вывел информацию о правах с помощью условного оператора и test

```
path="$1"
for i in ${path}/*
do
    echo "$i"

    if test -r $i
    then echo "Возможность чтения"
    fi
    if test -w $i
```

```

then echo "Возможность записи"
fi
if test -x $i
then echo "Возможность выполнения"
fi
done

```

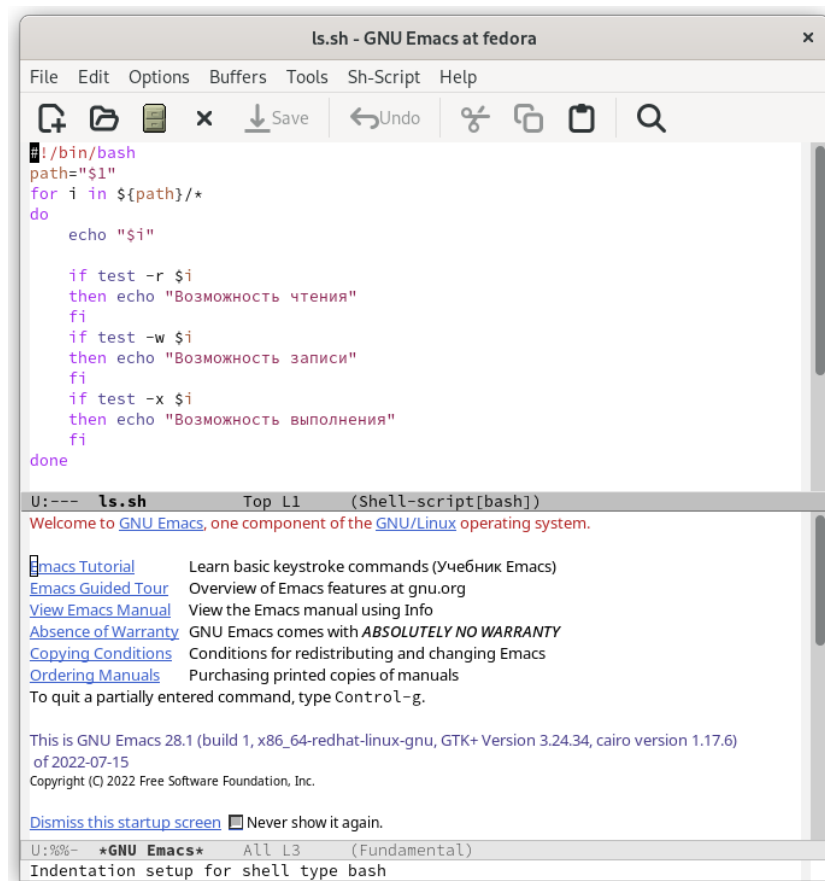


Figure 3.5: Написанный в Emacs скрипт

```
avkondratev@fedora:~  
/home/avkondratev/backup/backup.sh.bz2  
./ls.sh: строка 8: Возможность чтения: команда не найдена  
./ls.sh: строка 11: Возможность записи: команда не найдена  
./ls.sh: строка 14: Возможность выполнения: команда не найдена  
[avkondratev@fedora ~]$ emacs ls.sh  
[avkondratev@fedora ~]$ ./ls.sh ~/backup  
/home/avkondratev/backup/backup.sh.bz2  
Возможность чтения  
Возможность записи  
Возможность выполнения  
[avkondratev@fedora ~]$ cd backup  
[avkondratev@fedora backup]$ chmod -r zzzz  
[avkondratev@fedora backup]$ ./ls.sh ~/backup  
bash: ./ls.sh: Нет такого файла или каталога  
[avkondratev@fedora backup]$ cd ..  
[avkondratev@fedora ~]$ ./ls.sh ~/backup  
/home/avkondratev/backup/backup.sh.bz2  
Возможность чтения  
Возможность записи  
Возможность выполнения  
/home/avkondratev/backup/zzzz  
Возможность записи  
Возможность выполнения  
[avkondratev@fedora ~]$
```

Figure 3.6: Результат: информация по каждому файлу

4. Написал 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.3.7)(рис.3.8)

Передал в path путь до каталога, сделал сдвиг с помощью shift, чтобы не рассматривать путь как файл. Циклом for прошелся по аргументам(искомые форматы), инициализировал счетчик, еще одним циклом прошелся по файлам каталога, подставляя под регулярное выражение формат из первого цикла. Если файл существует, то счетчик увеличиваем на 1. В конце каждого внешнего цикла выводим количество найденных файлов

```
path="$1"  
shift  
for i in $@  
do  
    count=0
```

```

for j in ${path}/*${i}
do
if test -f "$j"
then
    let count=count+1
fi
done
echo "$count с расширением $i"
done

```

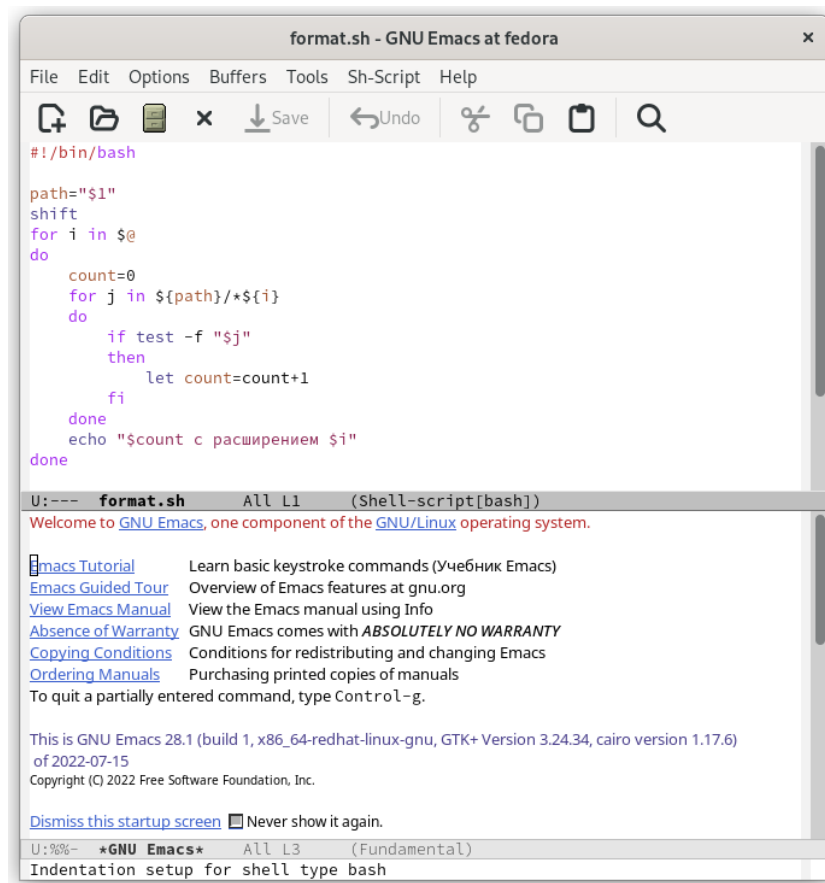
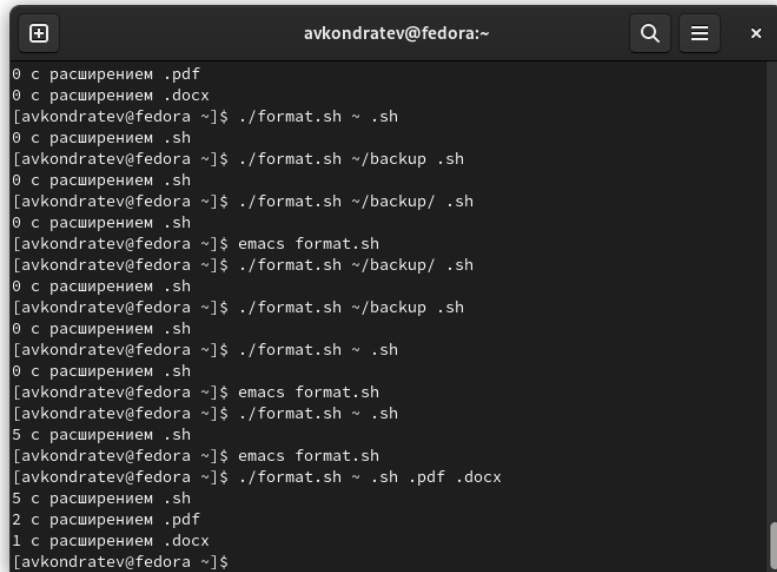


Figure 3.7: Написанный в Emacs скрипт



```
avkondratev@fedora:~  
0 с расширением .pdf  
0 с расширением .docx  
[avkondratev@fedora ~]$ ./format.sh ~ .sh  
0 с расширением .sh  
[avkondratev@fedora ~]$ ./format.sh ~/backup .sh  
0 с расширением .sh  
[avkondratev@fedora ~]$ ./format.sh ~/backup/ .sh  
0 с расширением .sh  
[avkondratev@fedora ~]$ emacs format.sh  
[avkondratev@fedora ~]$ ./format.sh ~/backup/ .sh  
0 с расширением .sh  
[avkondratev@fedora ~]$ ./format.sh ~/backup .sh  
0 с расширением .sh  
[avkondratev@fedora ~]$ ./format.sh ~ .sh  
0 с расширением .sh  
[avkondratev@fedora ~]$ emacs format.sh  
[avkondratev@fedora ~]$ ./format.sh ~ .sh  
5 с расширением .sh  
[avkondratev@fedora ~]$ emacs format.sh  
[avkondratev@fedora ~]$ ./format.sh ~ .sh .pdf .docx  
5 с расширением .sh  
2 с расширением .pdf  
1 с расширением .docx  
[avkondratev@fedora ~]$
```

Figure 3.8: Результат: количество файлов каждого формата

## 4 Выводы

Я изучил основы программирования в оболочке ОС UNIX/Linux. Научился писать небольшие командные файлы.

## 5 Контрольные вопросы

1. Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.
  - 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
  - BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна
2. POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
3. Переменные - с помощью команды mark  
Массивы - с помощью команды set
4. Команда read позволяет читать значения переменных со стандартного ввода  
Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению
5. Сложение(+), умножение(\*), деление(/), остаток от деления(%), вычитание(-)

6. Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки. Можно присваивать результаты условных выражений переменным, также как и использовать результаты арифметических вычислений в качестве условий
7. `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. Метасимволы - это символы, которые имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола
9. С помощью обратного слэша(`\`)
10. С помощью команды `touch`, указывая формат `sh`. Чтобы запустить, нужно перед названием файла в консоли написать (`./`)
11. С помощью `function` имя\_функции{тело функции}
12. С помощью команды `test`, которой можно проверить опцию `-d` для каталога или `-f` для файла



13. set - создание массива

unset - изъять переменную из программы

typeset - объявление переменной с возможностью задать ее тип

14. В консоли через пробел после названия. Обращение к ним происходит с помощью \$

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:-value} — если значение переменной name не определено, то оно будет заменено на указанное value