Операторы управления

В данной главе мы будем учиться размещать более одной команды в командной строке, используя для этого **операторы управления**. Также мы кратко обсудим связанные с этими операторами параметры (**$?**) и вопросы использования аналогичных операторам специальных символов (**&**).

Точка с запятой (;)

Вы можете разместить две и более команд в одной и той же строке, разделив эти команды с помощью символа точки с запятой ;. Командная оболочка будет исследовать строку команды до момента достижения символа точки с запятой. Все аргументы перед этим символом точки с запятой будут рассматриваться как аргументы, не относящиеся к команде, находящейся после символа точки с запятой. Все команды с наборами аргументов будут выполнены последовательно, причем командная оболочка будет ожидать завершения исполнения каждой из команд перед исполнением следующей команды.

```
[paul@RHELv4u3 ~]$ echo Hello
Hello
[paul@RHELv4u3 ~]$ echo World
```

```
World
[paul@RHELv4u3 ~]$ echo Hello ; echo World
Hello
World
[paul@RHELv4u3 ~]$
```

Амперсанд (&)

В том случае, если строка команды оканчивается символом амперсанда **&**, командная оболочка не будет ожидать завершения исполнения этой команды. Сразу же после ввода команды будет выведено новое приглашение командной оболочки, а сама команда будет исполняться в фоновом режиме. В момент завершения исполнения команды в фоновом режиме вы получите соответствующее сообщение.

```
[paul@RHELv4u3 ~]$ sleep 20 &
[1] 7925
[paul@RHELv4u3 ~]$
...ожидание в течение 20 секунд...
[paul@RHELv4u3 ~]$
[1]+  Done                    sleep 20
```

Технические подробности выполняющихся при использовании рассматриваемого оператора операций приведены в разделе, посвященном **процессам**.

Символ доллара со знаком вопроса ($?)

Код завершения предыдущей команды сохраняется в переменной командной оболочки с именем **$?**. На самом деле **$?** является параметром командной оболочки, а не ее переменной, так как вы не можете присвоить значение переменной **$?**.

```
paul@debian5:~/test$ touch file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
rm: невозможно удалить "file1": Нет такого файла или каталога
```

```
paul@debian5:~/test$ echo $?
1
paul@debian5:~/test$
```

Двойной амперсанд (&&)

Командная оболочка будет интерпретировать последовательность символов **&&** как **логический оператор "И"**. При использовании оператора **&&** вторая команда будет исполняться только в том случае, если исполнение первой команды успешно завершится (будет возвращен нулевой код завершения).

```
paul@barry:~$ echo первая команда && echo вторая команда
первая команда
вторая команда
paul@barry:~$ zecho первая команда && echo вторая команда
-bash: zecho: команда не найдена...
```

Во втором примере используется тот же принцип работы **логического оператора "И"**. Данный пример начинается с использования работоспособного варианта команды **cd** с последующим исполнением команды **ls**, после чего используется неработоспособный вариант команды **cd**, после которого команда **ls** не исполняется.

```
[paul@RHELv4u3 ~]$ cd gen && ls
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
[paul@RHELv4u3 gen]$ cd gen && ls
-bash: cd: gen: Нет такого файла или каталога
```

Двойная вертикальная черта (||)

Оператор || представляет **логическую операцию "ИЛИ"**. Вторая команда исполняется только тогда, когда исполнение первой команды заканчивается неудачей (возвращается ненулевой код завершения).

```
paul@barry:~$ echo первая команда || echo вторая
команда ; echo третья команда
первая команда
третья команда
paul@barry:~$ zecho первая команда || echo вторая команда
; echo третья команда
```

```
-bash: zecho: команда не найдена...
вторая команда
третья команда
paul@barry:~$
```

В следующем примере используется тот же принцип работы **логического оператора "ИЛИ"**.

```
[paul@RHELv4u3 ~]$ cd gen || ls
[paul@RHELv4u3 gen]$ cd gen || ls
-bash: cd: gen: Нет такого файла или каталога
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
```

Комбинирование операторов && и ||

Вы можете использовать описанные логические операторы "И" и "ИЛИ" для создания структур **условных переходов** в рамках строк команд. В данном примере используется команда **echo** для вывода информации о том, успешно ли отработала команда **rm**.

```
paul@laika:~/test$ rm file1 && echo Команда сработала! ||
echo Исполнение команды завершилось неудачей!
Команда сработала!
paul@laika:~/test$ rm file1 && echo Команда сработала! ||
echo Исполнение команды завершилось неудачей!
rm: невозможно удалить "file1": Нет такого файла или ка-
талога
Исполнение команды завершилось неудачей!
paul@laika:~/test$
```

Знак фунта (#)

Все написанное после **символа фунта (#)** игнорируется командной оболочкой. Это обстоятельство оказывается полезным при возникновении необходимости в написании **комментариев** в сценариях командной оболочки, причем **комментарии** ни коим образом не будут влиять на процесс исполнения команд или процесс раскрытия команд командной оболочкой.

```
paul@debian4:~$ mkdir test     # создаем директорию
paul@debian4:~$ cd test        #### переходим в эту дирек-
торию
```

```
paul@debian4:~/test$ ls       # пуста ли она ?
paul@debian4:~/test$
```

Экранирование специальных символов (\)

Символ обратного слэша \ позволяет использовать управляющие символы без их интерпретации командной оболочкой; процедура добавления данного символа перед управляющими символами называется **экранированием** символов.

```
[paul@RHELv4u3 ~]$ echo hello \; world
hello ; world
[paul@RHELv4u3 ~]$ echo hello\ \ \ world
hello   world
[paul@RHELv4u3 ~]$ echo экранирование \\\ \#\ \&\ \"\ \'
экранирование \ # & " '
[paul@RHELv4u3 ~]$ echo экранирование \\\?\*\"\'
экранирование \?*"'
```

Обратный слэш в конце строки

Строка команды, заканчивающаяся обратным слэшем, продолжается в следующей строке. Командная оболочка не будет интерпретировать символы перехода на новые строки и отложит исполнение операции раскрытия команды и ее исполнение до момента чтения новой строки команды без обратного слэша в конце.

```
[paul@RHEL4b ~]$ echo Данная строка команды \
> разделена на три \
> части
Данная строка команды разделена на три части
[paul@RHEL4b ~]$
```