Заявление по делу

Краткий обзор

```
случай <СЛОВО> в
 [(] <PATTERN1> ) <LIST1> ;; # или ;& или ;;& в Bash 4
 [(] <ШАБЛОН 2> ) <СПИСОК 2> ;;
 [(] <ШАБЛОН 3> | <ШАБЛОН 4> ) <СПИСОК 3-4> ;;
 ...
 [(] <Шаблон>) <Список> [;;]
esac
```

Описание

Оператор `case` -может выполнять команды на основе решения о сопоставлении с шаблоном. Слово `<WORD>` сопоставляется с каждым шаблоном `<PATTERNn>` , и при совпадении выполняется связанный список `<LISTn>` . Каждый список команд завершается `;;` . Это правило необязательно для самого последнего списка команд (т. Е. Вы можете опустить `;;` перед `esac`). Каждый `<PATTERNn>` отделяется от связанного `<LISTn>` с ним символом а `)` и необязательно предшествует а `(` .

В Bash 4 представлены два новых ограничителя действий. Классическое поведение `;;` заключается в том, чтобы выполнить только список, связанный с первым совпадающим шаблоном, а затем вырваться из `case` блока. `;&` Терминатор заставляет `case` также выполнить следующий блок, не проверяя его шаблон. `;;&` Оператор похож `;;` , за исключением того, что оператор case не завершается после выполнения связанного списка - Bash просто продолжает тестирование следующего шаблона, как если бы предыдущий шаблон не соответствовал. Используя эти терминаторы, `case` оператор может быть настроен для проверки на соответствие всем шаблонам или, например, для совместного использования кода между блоками.

Слово `<WORD>` расширяется с помощью *тильды*, *расширения* параметров *и переменных*; *арифметики*, *замены* команд *и процессов*; и *удаления кавычек*. **Разделение слов, скобки или расширение имени пути не выполняются**, что означает, что вы можете без проблем оставлять расширения без кавычек:

```
var="проверочное слово"

case $ var в
 ...
esac
```

Это похоже на поведение команды условного выражения ("новая тестовая команда") (также нет разделения слов для расширений).

В отличие от оператора C-case, выполняется только список соответствия и ничего больше. Если слову соответствует больше шаблонов, учитывается только первое совпадение. (Обратите **внимание** на комментарий об изменениях в Bash v4 выше.)

Шаблоны с несколькими `|` разделителями могут быть указаны для одного блока. Это совместимый с POSIX эквивалент `@(pattern-list)` конструкции extglob.

`case` Оператор - одна из самых сложных команд для четкого отступа, и люди часто спрашивают о наиболее "правильном" стиле. Просто делайте все возможное - существует множество вариантов стиля отступов `case` и нет реальной согласованной наилучшей практики.

Примеры

Еще один из моих глупых примеров...

```
printf '%s ' 'Какой фрукт ты любишь больше всего?'
прочитайте -${BASH_VERSION+e}r фрукты

дело $ fruit в
apple)
 эхо 'Мммм... Мне это нравится! '
 ;;
банан)
 эхо: "Хм, немного криво, нет?"
 ;;
апельсин | мандарин)
 echo $ 'Eeeks! Мне это не нравится!\НПО прочь! '
 выход 1
 ;;
 *)
эхо "Неизвестный фрукт - уверен, что он не токсичен?"
esac
```

Вот практический пример, показывающий общий шаблон, включающий `case` заявление. Если первый аргумент является одним из допустимого набора альтернатив, то выполните некоторые операции sysfs в Linux для управления профилем питания видеокарты. В противном случае покажите краткий обзор использования и выведите текущий профиль мощности и температуру графического процессора.

```
функция
# Set radeon power management clk {
        typeset  base=/sys/class/drm/card0/device
 [[ -r  $ {base}/hwmon/hwmon0/temp1_input && -r  $ {base}/power_profi
le ]] ||  возвращает  1

         случай  $ 1  в
 низкий | высокий | по умолчанию)
                        printf '%s \ n' "temp: $(<${base}/hwmon/hwmon
0/temp1_input)C" "старый профиль: $(<${base}/power_profile)"
                        echo "$1" > ${base}/power_profile
echo "новый профиль: $(<${base}/power_profile)"
                        ;;
                *)
                        echo "Использование: $FUNCNAME [ low | high |
default ]"
                        printf '%s \ n' "temp: $(<${base}/hwmon/hwmon
0/temp1_input)C" "текущий профиль: $(<$ {base}/power_profile)"
        esac
}
```

Шаблон для экспериментов с `case` логикой, показывающий общий код между используемыми блоками `;&` и оператор без короткого замыкания `;;&` :

```
#!/usr/bin/env bash

f() {
    local -a "$@"
    local x

 для x; сделайте
         case  $ x  в
             $ 1)
                 локальным  "$ x"'+=(1)' ;;&
            $2)
                 местный "$ x"'+=(2)' ;&
            $3)
                 местный "$ x"'+=(3)' ;;
            $ 1 | $ 2)
                 локальный  "$ x"'+= (4) '
        esac
         IFS=, локальный -a "$ x"'=("${x}: ${'"$ x"'[*]}") '
    сделано

     для x; сделать
         echo "$ {!x}"
    сделано
}

f a b c

# вывод:
 # a: 1,4
 # b: 2,3
 # c: 3
```

Соображения о переносимости

- `;;` В POSIX указан только разделитель.
- zsh и mksh используют оператор `;|` управления вместо оператора Bash `;;&` . Mksh `;;&` совместим с Bash (недокументирован).
- у ksh93 есть `;&` оператор, но нет `;;&` или эквивалент.
- ksh93, mksh, zsh и posh поддерживают исторический синтаксис, в котором вместо и могут использоваться открытые и закрытые фигурные `in esac` скобки : `case word { x) …; };` . Это похоже на альтернативную форму, которую Bash поддерживает для своих циклов for , но Bash не поддерживает этот синтаксис `case..esac` .

Смотрите также

- Условная конструкция POSIX case (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18_

Обсуждение

Р.У. Эмерсон II, 2013/02/25 03:19 ()

Было бы полезно получить дополнительную информацию о ШАБЛОНЕ. Например, в bash 4 источник ABS сообщает мне, что могут использоваться типы символов, например, "alnum". Но что, если кто-то хочет сопоставить переменное количество цифр, например? Комментарий в StackOverflow дал мне ответ.

http://stackoverflow.com/questions/4554718/patterns-in-case-statement-in-bash-scripting (http://stackoverflow.com/questions/4554718/patterns-in-case-statement-in-bash-scripting)

Кажется, что есть всевозможные возможности! Но ни один из них не задокументирован в разделе "дело" на моей странице man bash или в моем info bash. (В других местах есть документация для шаблонов в целом.)

.

Новые шаблоны bash 4 позволяют использовать case для проверки ответов. Мне нужна проверка, потому что непроверенный ответ может привести к сбою скрипта – например, если я присвою ответ целочисленной переменной, а ответ не является целым числом.

Я думаю, что это тот код, который мне нужен:

```
shopt -s extglob # Должен идти в основной строке, в идеале сразу
после shebang
# ...

прочитайте -p "Введите позицию сообщения (используйте знак для от
носительного положения): " tMPos

# Проверка ответа (tMPos) и обновление глобальной позиции сообщен
ия (vMPos):
#
случай "$ tMPos" в
 # Положительное целое число? Добавить к значению curr
 ([+]+([[:цифра:]]))
 vMPos = $ vMPos +10 # $ tReply
 ;;
 # Отрицательное целое число? Вычесть из значения curr
 ([-]+([[:цифра:]]))
 vMPos = $ vMPos-10 # $ tReply
 ;;
 # Целое число без знака? Заменить из curr значение
(+([[:цифра:]]))
 vMPos=10 # $tReply
 ;;
 # Другое: перехватите ошибку и верните ее обратно
 (*)
 echo "Ответ ($ tReply) не целое число"
esac
```

Я протестировал это, и это работает! – хотя gvim не очень нравится "]]))"!