The case statement

Synopsis

```
case <WORD> in
  [(] <PATTERN1> ) <LIST1> ;; # or ;& or ;;& in Bash 4
  [(] <PATTERN2> ) <LIST2> ;;
  [(] <PATTERN3> | <PATTERN4> ) <LIST3-4> ;;
  ...
  [(] <PATTERNn>) <LISTn> [;;]
esac
```

Description

The `case` -statement can execute commands based on a pattern matching decision. The word `<WORD>` is matched against every pattern `<PATTERNn>` and on a match, the associated list `<LISTn>` is executed. Every commandlist is terminated by `;;` . This rule is optional for the very last commandlist (i.e., you can omit the `;;` before the `esac`). Every `<PATTERNn>` is separated from it's associated `<LISTn>` by a `)` , and is optionally preceded by a `(` .

Bash 4 introduces two new action terminators. The classic behavior using `;;` is to execute only the list associated with the first matching pattern, then break out of the `case` block. The `;&` terminator causes `case` to also execute the next block without testing its pattern. The `;;&` operator is like `;;` , except the case statement doesn't terminate after executing the associated list - Bash just continues testing the next pattern as though the previous pattern didn't match. Using these terminators, a `case` statement can be configured to test against all patterns, or to share code between blocks, for example.

The word `<WORD>` is expanded using *tilde*, *parameter* and *variable expansion*; *arithmetic*, *command* and *process substitution*; and *quote removal*. **No word splitting, brace, or pathname expansion is done**, which means you can leave expansions unquoted without problems:

```
var="test word"

case $var in
  ...
esac
```

This is similar to the behavior of the conditional expression command ("new test command") (also no word splitting for expansions).

Unlike the C-case-statement, only the matching list and nothing else is executed. If more patterns match the word, only the first match is taken. (**Note** the comment about Bash v4 changes above.)

Multiple `|` -delimited patterns can be specified for a single block. This is a POSIX-compatable equivalent to the `@(pattern-list)` extglob construct.

The `case` statement is one of the most difficult commands to indent clearly, and people frequently ask about the most "correct" style. Just do your best - there are many variations of indenting style for `case` and no real agreed-upon best practice.

Examples

Another one of my stupid examples…

```
printf '%s ' 'Which fruit do you like most?'
read -${BASH_VERSION+e}r fruit

case $fruit in
    apple)
        echo 'Mmmmh... I like those!'
        ;;
    banana)
        echo 'Hm, a bit awry, no?'
        ;;
    orange|tangerine)
        echo $'Eeeks! I don\'t like those!\nGo away!'
        exit 1
        ;;
    *)
        echo "Unknown fruit - sure it isn't toxic?"
esac
```

Here's a practical example showing a common pattern involving a `case` statement. If the first argument is one of a valid set of alternatives, then perform some sysfs operations under Linux to control a video card's power profile. Otherwise, show a usage synopsis, and print the current power profile and GPU temperature.

```
# Set radeon power management
function clk {
        typeset base=/sys/class/drm/card0/device
        [[ -r ${base}/hwmon/hwmon0/temp1_input && -r ${base}/power_pr
ofile ]] || return 1

        case $1 in
                low|high|default)
                        printf '%s\n' "temp: $(<${base}/hwmon/hwmon0/
temp1_input)C" "old profile: $(<${base}/power_profile)"
                        echo "$1" >${base}/power_profile
                        echo "new profile: $(<${base}/power_profile)"
                        ;;
                *)
                        echo "Usage: $FUNCNAME [ low | high | default
]"
                        printf '%s\n' "temp: $(<${base}/hwmon/hwmon0/
temp1_input)C" "current profile: $(<${base}/power_profile)"
        esac
}
```

A template for experiments with `case` logic, showing shared code between blocks using `;&`, and the non-short-circuiting `;;&` operator:

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

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

    for x; do
        case $x in
            $1)
                local "$x"'+=(1)' ;;&
            $2)
                local "$x"'+=(2)' ;&
            $3)
                local "$x"'+=(3)' ;;
            $1|$2)
                local "$x"'+=(4)'
        esac
        IFS=, local -a "$x"'=("${x}: ${'"$x"'[*]}")'
    done

    for x; do
        echo "${!x}"
    done
}

f a b c

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

Portability considerations

- Only the `;;` delimiter is specified by POSIX.
- zsh and mksh use the `;|` control operator instead of Bash's `;;&`. Mksh has `;;&` for Bash compatability (undocumented).
- ksh93 has the `;&` operator, but no `;;&` or equivalent.
- ksh93, mksh, zsh, and posh support a historical syntax where open and close braces may be used in place of `in` and `esac`: `case word { x) …; };`. This is similar to the alternate form Bash supports for its for loops, but Bash doesn't support this syntax for `case..esac`.

See also

- POSIX case conditional construct (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18

Discussion

R.W. Emerson II, 2013/02/25 03:19 ()

More information about the PATTERN would be helpful. Under bash 4, for example, the ABS source tells me that character types – e.g., "alnum" – can be used. But what if one wants to match a variable number of digits, for example? A comment at StackOverflow gave me the answer.

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

It seems that there are all sorts of possibilities! But none of them are documented under "case" in my man bash page or my info bash. (There is documentation elsewhere for patterns in general.)

.

The new bash 4 patterns make it possible to use case to validate replies. I need validation, because an unvalidated reply can crash the script – if I assign the reply to an integer variable, for example, and the reply is not an integer.

I think this is the code I need:

```
shopt -s extglob  # Must go in main line, ideally just after the shebang
# ...

read -p "Enter the message position (use sign for relative position): " tMPos

# Validate the reply (tMPos) and update the global message position (vMPos):
#
case "$tMPos" in
  # Positive integer? Add to curr value
  ([+]+([[:digit:]]))
    vMPos=$vMPos+10#$tReply
    ;;
  # Negative integer? Subtract from curr value
  ([-]+([[:digit:]]))
    vMPos=$vMPos-10#$tReply
    ;;
  # Unsigned integer? Replace from curr value
  (+([[:digit:]]))
    vMPos=10#$tReply
    ;;
  # Other: Catch the error and throw it back
  (*)
    echo "Reply ($tReply) not an integer"
esac
```

I tested it, and it works! – though gvim doesn't much like the "]]))"!