The classic for-loop

Synopsis

```
for <NAME>; do
  <LIST>
done
```

```
for <NAME> in <WORDS>; do
  <LIST>
done
```

alternative, historical and undocumented syntax [1)]

```
for <NAME>; {
  <LIST>
}

for <NAME> in <WORDS>; {
  <LIST>
}
```

Description

For every word in `<WORDS>` , one iteration of the loop is performed and the variable `<NAME>` is set to the current word. If no " `in <WORDS>` " is present to give an own word-list, then the positional parameters (`"$@"`) are used (the arguments to the script or function). In this case (and only in this case), the semicolon between the variable name and the `do` is optional.

If you use the loop-variable inside the for-loop and it can contain spaces, you need to quote it, since normal word-splitting procedures apply.

Like all loops (both `for` -loops, `while` and `until`), this loop can be

- terminated (broken) by the `break` command, optionally as `break N` to break `N` levels of nested loops
- forced to immediately do the next iteration using the `continue` command, optionally as `continue N` analog to `break N`

Bash knows an alternative syntax for the `for` loop, enclosing the loop body in `{...}` instead of `do ... done` :

```
for x in 1 2 3
{
  echo $x
}
```

This syntax is **not documented** and should not be used. I found the parser definitions for it in 1.x code, and in modern 4.x code. My guess is that it's there for compatiblity reasons. This syntax is not specified by POSIX®.

Return status

The return status is the one of the last command executed in `<LIST>` or `0` (`TRUE`), if the item list `<WORDS>` evaluates to nothing (i.e.: "is empty"!).

Examples

Iterate over array elements

With some array syntax (see Arrays) you can easily "feed" the for-loop to iterate over all elements in an array (by mass-expanding all elements):

```
for element in "${myarray[@]}"; do
  echo "Element: $element"
done
```

Another way is to mass-expand all used indexes and access the array by index:

```
for index in "${!myarray[@]}"; do
  echo "Element[$index]: ${myarray[$index]}"
done
```

List positional parameters

You can use this function to test how arguments to a command will be interpreted and parsed, and finally used:

```
argtest() {
  n=1
  for arg; do
    echo "Argument $((n++)): \"$arg\""
  done
}
```

Loop through a directory

Since pathname expansion will expand all filenames to separate words, regardless of spaces, you can use the for-loop to iterate through filenames in a directory:

```
for fn in *; do
  if [ -h "$fn" ]; then
    echo -n "Symlink: "
  elif [ -d "$fn" ]; then
    echo -n "Dir: "
  elif [ -f "$fn" ]; then
    echo -n "File: "
  else
    echo -n "Unknown: "
  fi
  echo "$fn"
done
```

Stupid example, I know 😉

Loop over lines of output

To be complete: You can change the internal field separator (IFS) to a newline and thus make a for-loop iterating over lines instead of words:

```
IFS=$'\n'
for f in $(ls); do
  echo $f
done
```

This is just an example. In *general*

- it's not a good idea to parse `ls(1)` output
- the while loop (using the `read` command) is a better joice to iterate over lines

Nested for-loops

It's of course possible to use another for-loop as `<LIST>` . Here, counting from 0 to 99 in a weird way:

```
for x in 0 1 2 3 4 5 6 7 8 9; do
  for y in 0 1 2 3 4 5 6 7 8 9; do
    echo $x$y
  done
done
```

Loop over a number range

Beginning in Bash 4, you can also use "sequence expression" form of brace expansion syntax when looping over numbers, and this form does not create leading zeroes unless you ask for them:

```
# 100 numbers, no leading zeroes
for x in {0..99}; do
  echo $x
done
```

```
# Every other number, width 3
for x in {000..99..2}; do
  echo $x
done
```

WARNING: the entire list is created before looping starts. If your list is huge this may be an issue, but no more so than for a glob that expands to a huge list.

Portability considerations

See also

- The C-style for-loop

1)

http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xcu_chap02.html#tag_23_02_09_12 (http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xcu_chap02.html#tag_23_02_09_12)

Discussion

syntax/ccmd/classic_for.txt Last modified: 2017/01/19 21:02 by 4dummies