[[commands:builtin:declare]]

The declare builtin command

Synopsis

```
declare [-aAfFgilnrtux] [-p] [NAME[=VALUE] ...]

# obsolete typeset synonym
typeset [-aAfFgilnrtux] [-p] [NAME[=VALUE] ...]
```

Description

`declare` is used to display or set variables along with variable attributes. When used to display variables/functions and their value, the output is re-usable as input for the shell.

If no `NAME` is given, it displays the values of all variables or functions when restricted by the `-f` option.

If `NAME` is followed by `=VALUE`, `declare` also sets the value for a variable.

When used in a function, `declare` makes `NAMEs` local variables, unless used with the `-g` option.

Don't use it's synonym `typeset` when coding for Bash, since it's tagged as obsolete.

Options

Below, `[-+]X` indicates an attribute, use `-X` to set the attribute, `+X` to remove it.

Option	Description
`[-+]a`	make NAMEs indexed arrays (removing with `+a` is valid syntax, but leads to an error message)
`[-+]A`	make NAMEs associative arrays
`[-+]c`	**Undocumented** convert NAMEs to "capcase" on assignment (makes the first letter upper-case and the rest lower). Requires Bash built with `-DCASEMOD_CAPCASE`

Option	Description
`-f`	restrict action or display to function names and definitions (removing with `+f` is valid syntax, but leads to an error message)
`-F`	restrict display to function names only (plus line number and source file when debugging)
`-g`	create global variables when used in a shell function; otherwise ignored (by default, `declare` declares local scope variables when used in shell functions)
`[-+]i`	make NAMEs have the "integer" attribute
`[-+]l`	convert NAMEs to lower case on assignment (makes sure the variable contains only lower case letters)
`[-+]n`	make NAME a reference to the variable named by its value. Introduced in Bash 4.3-alpha. `${!NAME}` reveals the reference variable name, VALUE. Use `unset -n NAME` to unset the variable. (`unset -v NAME` unsets the VALUE variable.) Use `[[ -R NAME ]]` to test if NAME has been set to a VALUE, another variable's name.
`-p`	display the attributes and value of each NAME
`[-+]r`	make NAMEs readonly (removing with `+r` is valid syntax, but not possible)
`[-+]t`	make NAMEs have the "trace" attribute (effective only for functions)
`[-+]u`	convert NAMEs to upper case on assignment (makes sure the variable contains only upper case letters)
`[-+]x`	make NAMEs exported

Return status

Status	Reason
0	no error
!= 0	invalid option
!= 0	invalid variable name given
!= 0	attempt to **define** a function using `-f`
!= 0	assignment to a readonly variable
!= 0	removing the readonly-attribute from a readonly variable

Status	Reason
!= 0	assignment to an array variable without the compound assignment syntax (`array=(…)`)
!= 0	attempt to use `+a` to "destroy" an array
!= 0	attemt to display a non-existent function with `-f`

Notes

Unix shells offer very few datatypes. Bash and some other shells extend this by allowing "attributes" to be set on variable names. The only attributes specified by POSIX are `export` and `readonly` , which are set by their own dedicated builtins. Datatypes in bash have a few other interesting capabilities such as the ability to modify data on assignment.

Examples

Display defined functions

`declare -f` can be used to display all defined functions…

```
$ declare -f
foo ()
{
    echo "FOO is BAR"
}
world ()
{
    echo "Hello World!"
}
```

…or just a specific defined function.

```
$ declare -f foo
foo ()
{
    echo "FOO is BAR"
}
```

Nameref

Bash 4.3 adds a new way to indirectly reference variables. `typeset -n` or `declare -n` can be used to make a variable indirectly refer to another. In Bash, the lvalue of the assignment given to `typeset -n` or `declare -n` will refer to the variable whose name is expanded on the RHS.

`typeset -n` is used in the example below. See notes below.

```
# Sum a set of arrays and assign the result indirectly, also printing
each intermediary result (without portability workarounds)
# sum name arrname [ arrname ... ]
function sum {
    typeset -n _result=$1 _arr
    typeset IFS=+
    _result=0
    for _arr in "${@:2}"; do                        # Demonstrate the
special property of "for" on a nameref.
        (( _result += ${_arr[*]} ))
        printf '%s = %d\n' "${!_result}" "$_result" # Demonstrate the
special property of ${!ref} on a nameref.
    done
}

a=(1 2 3) b=(6 5 4) c=(2 4 6)
sum total a b c
printf 'Final value of "total" is: %d\n' "$total"
```

`typeset -n` is currently implemented in ksh93, mksh, and Bash 4.3. Bash and mksh's implementations are quite similar, but much different from ksh93's. See Portability considerations for details. ksh93 namerefs are much more powerful than Bash's.

Portability considerations

- `declare` is not specified by POSIX®
- `declare` is unique to Bash and totally non-portable with the possible exception of Zsh in Bash compatibility mode. Bash marks the synonym `typeset` as obsolete, which in Bash behaves identically to `declare`. All other Korn-like shells use `typeset`, so it probably isn't going away any time soon. Unfortunately, being a non-standard builtin, `typeset` differs significantly between shells. ksh93 also considers `typeset` a special builtin, while Bash does not - even in POSIX mode. If you use `typeset`, you should attempt to only use it in portable ways.
- **todo** nameref portability…

See also

- Arrays
- The readonly builtin command
- The unset builtin command
- declaration commands (http://austingroupbugs.net/view.php?id=351) will change the behavior of certain builtins such as `export` in the next version of POSIX.

Discussion

commands/builtin/declare.txt Last modified: 2022/11/09 01:48 by fgrose

This site is supported by Performing Databases - your experts for database administration

Bash Hackers Wiki

Except where otherwise noted, content on this wiki is licensed under the following license:
GNU Free Documentation License 1.3