The echo builtin command

Synopsis

```
echo [-neE] [arg ...]
```

Description

`echo` outputs it's args to stdout, separated by spaces, followed by a newline. The return status is always `0`. If the shopt option `xpg_echo` is set, Bash dynamically determines whether echo should expand escape characters (listed below) by default based on the current platform. `echo` doesn't interpret `--` as the end of options, and will simply print this string if given.

Options

Option	Description
`-n`	The trailing newline is suppressed.
`-e`	Interpretation of the following backslash-escaped characters (below) is enabled.
`-E`	Disables the interpretation of these escape characters, even on systems where they are interpreted by default.

Escape sequences

Escape	Description
`\a`	alert (bell)
`\b`	backspace
`\c`	suppress further output
`\e`	
`\E`	an escape character

Escape	Description
`\f`	form feed
`\n`	new line
`\r`	carriage return
`\t`	horizontal tab
`\v`	vertical tab
`\\`	backslash
`\0nnn`	the eight-bit character whose value is the octal value nnn (zero to three octal digits)
`\xHH`	the eight-bit character whose value is the hexadecimal value HH (one or two hex digits)
`\uHHHH`	the Unicode (ISO/IEC 10646) character whose value is the hexadecimal value HHHH (one to four hex digits)
`\UHHHHHHHH`	the Unicode (ISO/IEC 10646) character whose value is the hexadecimal value HHHHHHHH (one to eight hex digits)

Examples

Portability considerations

- `echo` is a portability train wreck. No major shell follows POSIX completely, and any shell that attempts to do so should be considered horribly broken. SUSv4 (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/echo.html#tag_20_37) specifies that `echo` **shall not** include any options. Further, it specifies that the behavior of `-n` as a first argument shall be determined by the implementation, unless XSI is followed, in which case `-n` is always treated as a string, and backslash escapes are interpreted by default. `dash` has the misfeature of following this and interpreting escapes by default, but includes a `-n` feature for suppressing newlines nevertheless.

  In practice, if you're able to assume a korn-like shell including bash, mksh, or zsh, `echo` when used in simple cases is generally reliable. For example, in the very common situation in which echo is supplied with a single argument and whose output is to have a newline appended, using `echo` is considered common practice.

- **Never use options to `echo` ! *Ever*!** Any time you feel tempted to use `echo -e`, `-n`, or any other special feature of echo, **use printf instead!** If portability is a requirement, you should consider using `printf` *exclusively* and just ignore that `echo` even exists. If you must use `echo -e` and refuse to use `printf`, it is

usually acceptable to use `echo $'…'` if targeting only shells that support this special quoting style.

- `ksh93` has a `print` command, which if coding specifically for `ksh93` should be preferred over `echo`. printf still includes most of the functionality of both, and should usually be the most preferred option.

See also

- The printf command
- http://cfajohnson.com/shell/cus-faq.html#Q0b (http://cfajohnson.com/shell/cus-faq.html#Q0b)
- http://www.in-ulm.de/~mascheck/various/echo+printf/ (http://www.in-ulm.de/~mascheck/various/echo+printf/)

Discussion

M. Antler (http://www.example.com), 2016/05/01 04:12 ()

1. Use -E.
2. Quote your variables properly.
3. Use only 1 non-option argument.

```
$ a='-e' ; b='1\t2'
$ echo -E $a $b
1       2
$ echo -E "$a $b"
-e 1\t2
$
```

commands/builtin/echo.txt Last modified: 2012/05/26 11:14 by ormaaj