Obsolete and deprecated syntax

This (incomplete) page describes some syntax and commands considered obsolete by some measure. A thorough discussion of the rationale is beyond the scope of this page. See the portability page for a discussion on portability issues.

This first table lists syntax that is tolerated by Bash but has few if any legitimate uses. These features exist mostly for Bourne, csh, or some other backward compatibility with obsolete shells, or were Bash-specific features considered failed experiments and deprecated or replaced with a better alternative. These should be irrelevant to most everyone except maybe code golfers. New scripts should never use them. None of the items on this list are specified by the most current version of POSIX, and some may be incompatible with POSIX.

Syntax	Replacement	Description
`&>FILE` and `>&FILE`	`>FILE 2>&1`	This redirection syntax is `>FILE 2>&1` and origin C Shell. The latter form i uncommon and should r used, and the explicit for separate redirections is p over both. These shortcu contribute to confusion a copy descriptor because

Syntax	Replacement	Description

		is unclear. They also intr parsing ambiguity, and c POSIX. Shells without thi treat `cmd1 &>file cmc` "background `cmd1` and execute `cmd2` with its st redirected to `file` ", wh correct interpretation of t expression. See: redirec
		`$ { bash; dash </`
		`; } <<<'echo foo>/`
		`&>/dev/fd/2 echo b`
		`foo echo bar`
		`bar`

`$[EXPRESSION]`	`$((EXPRESSION))`	This undocumented synt completely replaced by t conforming arithmetic ex `$((EXPRESSION))` . It is unimplemented almost e except Bash and Zsh. Se arithmetic expansion. So discussion (http://lists.gnu.org/archiv bash/2012-04/msg0003
`COMMAND |& COMMAND`	`COMMAND 2>&1 | COMMAND`	This is an alternate pipel derived from Zsh. Officia considered deprecated b I highly discourage it. It c the list operator used for creation in most Korn sh has confusing behavior. is redirected first like an pipe, while the stderr is a redirected last – after oth preceding the pipe opera it's pointless syntax bloa explicit redirect instead.
`function NAME() COMPOUND-CMD`	`NAME() COMPOUND-CMD` or `function NAME { CMDS; }`	This is an amalgamation the Korn and POSIX styl definitions - using both th `function` keyword and parentheses. It has no u purpose and no historica reason to exist. It is not s POSIX. It is accepted by mksh, zsh, and perhaps

Syntax	Replacement	Description
		Korn shells, where it is tr identical to the POSIX-st It is not accepted by AT& should never be used. S table for the `function` Bash doesn't have this fe documented as expressl deprecated.
`for x; { …;}`	`do` , `done` , `in` , `esac` , etc.	This undocumented synt the `do` and `done` reser with braces. Many Korn s support various permuta syntax for certain compo commands like `for` , `ca` `while` . Which ones and details like whether a new semicolon are required v `for` works in Bash. Nee say, don't use it.

This table lists syntax that is specified by POSIX (unless otherwise specified below), but has been superseded by superior alternatives (either in POSIX, Bash, or both), or is highly discouraged for other reasons such as encouraging bad practices or dangerous code. Those that are specified by POSIX may be badly designed and unchangeable for historical reasons.

Syntax	Replacement	
Unquoted expansions, Word splitting, and Pathname expansion (globbing)	Proper quoting (http://mywiki.wooledge.org/Quotes), Ksh/Bash-style arrays, The "$@" expansion, The read builtin command	*Quotir* mistak carriec compl previo impor time fr most r shells perfor **expar** This m variab depen differe side-e

Syntax	Replacement	
		of files globbi (witho charac evalua (http:/ possib shell a standa writter excep signific Pitfalls fall un *with fo (http:/*
`` `COMMANDS` ``	`$(COMMANDS)`	This is comm `$(CO` but the is unfo style c implen some) compa Backti escap wild ar See: *W (http:/*
`[ EXPRESSION ]` and `test EXPRESSION`	`[[ EXPRESSION ]]`	`test` evalua identic expres Ksh/Ba analog vs `[` argum one ex numbe use th comm need t reason and co See: T

Syntax	Replacement	See: t *differe* *(http:/*

`set -e`, `set -o errexit` and the `ERR` trap	proper control flow and error handling	`set`
		be fata
		only d
		produ
		high-a
		of this
		find th
		Think
		`throv`
		better
		entirel
		when
		Conve
		experi
		manda
		Becau

Syntax	Replacement	
		the er the `DI` and b comm Most I've se `pipe` POSIX featur debug **The `s` and fa than a** rely or to take **exact** *set -o (http:/* http://v (http:/
`set -u` or `set -o nounset`	Proper control flow and error handling	`set` variab `e` , it b from t status non-tr hacks might guarar error v (http:/ (http:/ http://v ulm.de (http:/ ulm.de

Syntax	Replacement	
		Appar *How d define (http:/* to prop `-u` .
`${var?msg}` or `${var:?msg}`	Proper control flow and error handling	Like s which enviro null. It the op create test fo techni (http:/ handle This e `set` also a decep `bash "${p f -v bash`

This table lists features that are used only if you have a specific reason to prefer it over another alternative. These have some legitimate uses if you know what you're doing, such as for those with specific portability requirements, or in order to make use of some subtle behavioral differences. These are frequently (mis)used for no reason. Writing portable scripts that go outside of POSIX features requires knowing how to account for many (often undocumented) differences across many shells. If you do happen to know what you're doing, don't be too surprised if you run across someone telling you not to use these.

Syntax	Replacement	Description
`function NAME { CMDS; }`	`NAME() COMPOUND-CMD`	This is the ksh form of function defir created to extend the Bourne and P form with modified behaviors and additional features like local variable idea was for new-style functions to b analogous to regular builtins with the environment and scope, while POSI functions are more like special builti `function` is supported by almost e ksh-derived shell including Bash and but isn't specified by POSIX. Bash tr function styles the same, but this is unusual. `function` has some prefe characteristics in many ksh variants

Syntax	Replacement	Description
		making it more portable for scripts th non-POSIX extensions by some mea If you're going to use the `functior` keyword, it implies that you're either targeting Ksh specifically, or that you detailed knowledge of how to compe for differences across shells. It shou always be used consistently with `typeset`, but never used with `dec` or `local`. Also in ksh93, the braces not a command group, but a require of the syntax (unlike Bash and other shell function definitions
`typeset`	`declare`, `local`, `export`, `readonly`	This is closely related to the above, should often be used together. `typ` exists primarily for `ksh` compatibilit marked as "deprecated" in Bash (th don't entirely agree with this). This r some sense, because future compa can't be guaranteed, and any comp at all, requires understanding the nc POSIX features of other shells and t differences. Using `declare` instea `typeset` emphasizes your intentio "Bash-only", and definitely breaks everywhere else (except possibly zs you're lucky). The issue is further complicated by Dash and the Debiar (http://www.debian.org/doc/debian-policy/ch-files.html#s-scripts) require for a `local` builtin, which is itself n entirely compatible with Bash and ot shells.
`let 'EXPR'`	`((EXPR))` or `[ $((EXPR)) -ne 0 ]`	`let` is the "simple command" varia arithmetic evaluation command, whi takes regular arguments. Both `let` `((expr))` were present in ksh88

Syntax	Replacement	Description
		`((expr))` were present in ksh88, everything that supports one should support the other. Neither are POSI compound variant is preferable bec doesn't take regular arguments for wordsplitting and globbing, which m safer and clearer. It is also usually f especially in Bash, where compoun commands are typically significantly Some of the (few) reasons for using are detailed on the let page. See ar evaluation compound command
`eval`	Depends. Often code can be restructured to use better alternatives.	`eval` is thrown in here for good m as sadly it is so often misused that of `eval` (even the rare clever one) immediately dismissed as wrong by experts, and among the most imme solutions abused by beginners. In r there are correct ways to use `eval` even cases in which it's necessary, sophisticated shells like Bash and K `eval` is unusual in that it is less fre appropriate in more feature-rich she in more minimal shells like Dash, w is used to compensate for more limi If you find yourself needing `eval` t frequently, it might be a sign that yo either better off using a different lan entirely, or trying to borrow an idiom some other paradigm that isn't well to the shell language. By the same t there are some cases in which work hard to avoid `eval` ends up adding of complexity and sacrificing all port Don't substitute a clever `eval` for something that's a bit "too clever", j

Syntax	Replacement	Description
		avoid the `eval`, yet, take reasonab measures to avoid it where it is sens do so. See: The eval builtin commar Eval command and security issues (http://mywiki.wooledge.org/BashFA

See also

- Non-portable syntax and command uses
- Bash changes
- Greg's BashFAQ 061: List of essential features added (with the Bash version tag) (http://mywiki.wooledge.org/BashFAQ/061)
- Bash <-> POSIX Portability guide with a focus on Dash (http://mywiki.wooledge.org/Bashism)
- http://mywiki.wooledge.org/BashPitfalls (http://mywiki.wooledge.org/BashPitfalls)

Discussion

scripting/obsolete.txt Last modified: 2019/08/30 16:01 by ersen