AWK

December 27, 2025

Canine-Table

POSIX Nexus serves as a comprehensive cross-language reference hub that explores the implementation and behavior of POSIX-compliant functionality across a diverse set of programming environments. Built atop the foundational IEEE Portable Operating System Interface (POSIX) standards, this project emphasizes compatibility, portability, and interoperability between operating systems.

Abstract

Contents

I Shell Modules

I The Argument Registry

$$
\begin{aligned}
A_g &= \text{leader aliases} + \text{member aliases} \\
S_g &= 1 + A_g && \text{(canonical leader slot + alias slots)} \\
S_{\text{groups}} &= \sum_g S_g = \sum_g (1 + A_g) \\
S_{\text{base}} &= 12 && \text{(fixed non-group stride slots)} \\
S_{\text{total}} &= S_{\text{base}} + S_{\text{groups}}
\end{aligned}
$$

I The Argument Tokenizer

Shell Option Tokenizer Arguments

- D1 → Option string to tokenize (raw input)
- N → Debug level:
  - 0 → silent
  - >0 → errors only
  - >1 → warnings
  - >2 → verbose audit
  - >3 → mapping summary
  - >4 → full dump
- V → Array to populate with parsed flags, keywords, arrays, groups
- D2 → Separator of separators (defaults to comma)
- D3 → List of separator characters (key, alias, array, group, etc.)

```
function nx_sh_opts(D1, N, V, D2, D3)
```

Usage essentials

- **Flags** Standalone options, stride slot 4
- **Keywords** Key/value pairs, stride slot 1
- **Flag arrays** Appendable flags, stride slot 7
- **Keyword arrays** Appendable keywords, stride slot 10
- **Groups** Leaders + members, stride slot 13+
- **Aliases** Use '&' immediately after option
- **Separators** Use non-alphanumeric single characters
- **Logging** Set N for verbosity level

I Option Separators D3

Variable Positions in trk

Index	Default	Variable	Meaning
–	','	ds	Default delimiter string
1	':'	ks	Key separator
2	'&'	als	Alias / altname separator
3	'@'	fas	Appendable flag array separator
4	'#'	kas	Appendable keyword array separator
5	'<'	go	Begin group marker
6	'>'	gc	End group marker
7	' '	lo	Begin or continue long option mode
8	';'	lc	End long option mode

Short vs Long Option Group Leaders

- **Short form** Input: ‘alpha<beta␣gamma>’ → Flags = a, l, p, h; Group leader = a; Members = b, e, t, a, gamma
- **Long form** Input: ‘␣alpha<beta␣gamma>’ → Group leader = alpha; Members = beta, gamma
- **Key distinction** Leading space (‘lo’) preserves token as long option; without space, token explodes into flags and first flag becomes leader

Defaults for Option Groups

- **Group 1** ‘#’ keyword arrays
- **Group 2** ‘@’ flag arrays
- **Group 3** ‘:’ keywords
- **Group 4** ‘␣’ flags
- **Alias Requirement** Each group requires one alias slot
- **Stride Variable** Denoted by S
- **Group Count** Denoted by G
- **Alias Slots** Denoted by A

I Stride mappings

Variables

- **D1** Options string
- **go, gc** Group open/close markers
- **als** Alias symbol
- **D2** Runtime group stride contribution (S_{groups})
- **strde** Final array stride (S_{total})

$$
\begin{aligned}
A_g &= \text{leader aliases} + \text{member aliases} \\
S_g &= 1 + A_g \quad \text{(canonical leader slot + alias slots)} \\
S_{\text{groups}} &= \sum_g S_g = \sum_g (1 + A_g) \\
S_{\text{total}} &= 12 + S_{\text{groups}}
\end{aligned}
$$

Anchors spaced by S_{total}

Category	Base index	Walk by
Keywords	1	$+ S_{\text{total}}$
Flags	4	$+ S_{\text{total}}$
Flag arrays	7	$+ S_{\text{total}}$
Keyword arrays	10	$+ S_{\text{total}}$
Groups (triplets)	13	$+ 3$ per group header, values spaced by S_{total}

I Examples

Sample inputs and outputs

- **Input** ⇝ ‘␣alpha&al<beta&b␣gamma&g>’
- **Output** ⇝ Group leader = al; Members = b, g; Aliases = alpha, beta, gamma
- **Input** ⇝ ‘␣opt1@opt2#opt3’
- **Output** ⇝ Flag array = opt1, keyword array = opt2, flag = opt3

I The Argument Parser

I Option Grammar

Flag semantics

- **-n** Flag true $\langle nx : true/\rangle$
- **-N** Flag false $\langle nx : false/\rangle$
- **--No** Alias of Not, false
- **--Not** Canonical flag, false

Keyword semantics

- **Gobble** Consume next token
- **=** Explicit set
- **= (empty)** Clear value
- **+** Append/push
- **+n** Repeat append n times
- **-** Pop/remove
- **-0** Global substitution (gsub)

Case polarity

- **Lowercase** Affirmative action
- **Uppercase** Opposite action

I Examples

Sample inputs and outputs

- **Input** ⇝ ‘␣n&No&Not’
- **Output** ⇝ Flag = Not; Aliases = n, No; -n → <nx:true/>, --No → <nx:false/>
- **Input** ⇝ ‘␣key1:key2’
- **Output** ⇝ Keyword family = key1; Gobble next token as value; Alias = key2
- **Input** ⇝ ‘␣arr1@arr2’
- **Output** ⇝ Flag array = arr1; Keyword array = arr2
- **Input** ⇝ ‘--value=hello’
- **Output** ⇝ Keyword set; value = “hello”
- **Input** ⇝ ‘--value+=world’
- **Output** ⇝ Keyword append; value = previous + “world”
- **Input** ⇝ ‘--value-0=foo’
- **Output** ⇝ Keyword pop with gsub; remove all “foo” from value
- **Input** ⇝ ‘--Flag’
- **Output** ⇝ Boolean flag; uppercase → <nx:false/>
- **Input** ⇝ ‘--flag’
- **Output** ⇝ Boolean flag; lowercase → <nx:true/>

I Option Action Dispatcher

Option Preprocessor Arguments

- V1 — Argument vector (raw tokens)
- V2 — Hashmap of option values (mutable registry)
- V3 — Action descriptors (fs, fsa, fsr, separators)
- D1 — Group leader
- D2 — Alias used
- D3 — Suffix or group marker
- N1 — Category code
- N2 — Debug level
- N3 — Group index
- N4 — Current argument index

```
1  function nx_sh_opts_pre(V1, V2, V3, D1, D2, D3, N1, N2, N3, N4)
```

Option Preprocessor — Internal Semantics

- **Purpose** — Dispatch option to correct action handler based on category and operator
- **Mechanism** — Resolves polarity, separators, prefixes, and invokes fsa/fsr/fs/gobble helpers
- **Category Behavior** — Keyword, Flag, Flag Array, Keyword Array each adjust parameters before dispatch
- **Return** — Updated argument index $N4$
- **Use Case** — Core engine for option mutation; called once per parsed option

Semantic Examples

- Keyword `--opt foo`, `--opt=foo`, `--opt+=foo`
- Flag `-f`, `--Flag`, `--Flag+=true`
- Flag Array `--arr foo`, `--arr+=foo`, `--arr-=foo`
- Keyword Array `--opt bar`, `--opt+=bar`, `--opt-=bar`

I Action: Push (fsa)

Push Action Arguments (Internal Use)

- V — Hashmap of option values (mutable registry)
- D1 — Value to append (string)
- D2 — Separator or polarity marker
- D3 — Existing value (string)
- D4 — Option key
- N1 — Category code (1=kwd, 4=flag, 7=flag array, 10=kwd array)
- N2 — Repeat count
- B — Case polarity (boolean sentinel)

```
function nx_sh_opts_pre_act_fsa(V, D1, D2, D3, D4, N1, N2, B)
```

Push Action — Internal Semantics

- **Purpose** → Append value $D1$ to existing option $D4$, respecting category and polarity
- **Mechanism** → Flags use boolean polarity; keywords and arrays use string append via `nx_append_str`
- **Category Behavior** → Flag → boolean append; Keyword → string append; Arrays → separator-aware append
- **Return** → Updated registry entry $V[D4]$
- **Use Case** → Invoked by the option preprocessor when encountering += actions

Semantic Examples

- **Keyword** → --Opt+=foo → enqueue “foo” to existing value
- **Flag** → -f+ → boolean append using polarity
- **Flag Array** → --arr+=bar → append “bar” with array separator
- **Keyword Array** → --opt+=baz append “opt baz” as array element

I Action: Pop (fsr)

Pop Action Arguments (Internal Use)

- `V` ⟿ Hashmap of option values (mutable registry)
- `D1` ⟿ Value used for pop (string)
- `D2` ⟿ Separator or polarity marker
- `D3` ⟿ Existing value (string)
- `D4` ⟿ Option key
- `N1` ⟿ Category code (1=keyword, 4=flag, 7=flag array, 10=keyword array)
- `N2` ⟿ Count of removals
- `B` ⟿ Case polarity (boolean sentinel)

```
function nx_sh_opts_pre_act_fsr(V, D1, D2, D3, D4, N1, N2, B)
```

Pop Action — Internal Semantics

- **Purpose** ⟿ Remove occurrences of $D1$ or array elements from option $D4$
- **Mechanism** ⟿ Keywords use `nx_reap_str`; arrays use `nx_reap_str_match`; flags use polarity inversion
- **Category Behavior** ⟿ Keyword → substring pop; Flag → boolean pop; Arrays → element pop with separator
- **Return** ⟿ Updated registry entry $V[D4]$
- **Use Case** ⟿ Triggered when parser encounters a `-=` or `-n=` action

Semantic Examples

- **Keyword** --Opt-=foo → remove "foo" from value from the start
- **Keyword (count)** --opt-2=foo → remove two occurrences from the end
- **Flag** -F- → boolean pop (invert or clear)
- **Flag Array** --arr-=bar → remove "bar" from array
- **Keyword Array** --opt-=baz → remove "opt baz" element

I Action: Set (fs)

Set Action Arguments (Internal Use)

- V Hashmap of option values (mutable registry)
- D1 Value to assign (string)
- D2 Option key
- N Category code (1=keyword, 4=flag, 7=flag array, 10=keyword array)
- B Case polarity (boolean sentinel)

```
1 function nx_sh_opts_pre_act_fs(V, D1, D2, N, B)
```

Set Action — Internal Semantics

- **Purpose** Assign value $D1$ directly to option $D2$
- **Mechanism** Flags use boolean fallback when value is empty; other categories assign raw string
- **Category Behavior** Flag → assign or toggle; Keyword/Arrays → direct assignment
- **Return** Updated registry entry $V[D2]$
- **Use Case** Triggered when parser encounters a plain = action

Semantic Examples

- **Keyword** `--opt=foo` → set value to “foo”
- **Keyword (clear)** `--opt=` → clear value
- **Flag** `--Flag=` → boolean fallback based on case
- **Flag Array** `--arr=foo` → replace entire array with “foo”
- **Keyword Array** `--opt=bar` → replace array with “bar”

I Action: Gobble (no operator)

Gobble Action Arguments (Internal Use)

- V1 Hashmap of option values (mutable registry)
- V2 Argument vector (raw tokens)
- D1 Prefix for keyword arrays
- D2 Option key
- N1 Category code
- N2 Current argument index

```
function nx_sh_opts_pre_act(V1, V2, D1, D2, N1, N2)
```

Gobble Action — Internal Semantics

- **Purpose** Consume the next token from the argument vector and assign it to option $D2$
- **Mechanism** Keyword arrays prepend their key; other categories store raw token
- **Category Behavior** Keyword → gobble next token; Flag → never gobbles; Arrays → gobble with or without prefix
- **Return** Updated argument index $N2$
- **Use Case** Triggered when no operator (=, +=, -=) is present

Semantic Examples

- **Keyword** `--opt foo` → value becomes “foo”
- **Keyword Array** `--opt bar` → value becomes “opt bar”
- **Flag** `-f` → handled by boolean logic, not gobble
- **Flag Array** `--arr baz` → append “baz”