

# Posix-Nexus AWK



Canine-Table

March 24, 2025

## Contents

<b>1 Algorithms</b>	<b>III</b>
<b>2 Boolean</b>	<b>III</b>
<b>3 Math</b>	<b>X</b>
<b>4 Strings</b>	<b>X</b>
<b>5 Type Validation</b>	<b>X</b>
<b>6 Numbers Base</b>	<b>XII</b>
<b>7 Internet</b>	<b>XII</b>
<b>8 Miscellaneous</b>	<b>XII</b>
<b>9 Standard Output</b>	<b>XII</b>
<b>10 Structures</b>	<b>XII</b>

# 1 Algorithms

## 2 Boolean

### Boolean Operations

The following functions provide utilities for logical and comparative operations, enabling versatile Boolean checks across various conditions.

- ➔ **NOT\_\_(D)**: Returns the logical NOT of D.
- ➔ **NULL\_\_(D)**: Returns the logical NOT of D (equivalent to **NOT\_\_**).
- ➔ **FULL\_\_(D)**: Determines whether D is full (non-empty).
- ➔ **TRUE\_\_(D, B)**: Returns 1 if D is full or valid based on B.
- ➔ **FALSE\_\_(D, B)**: Returns the logical NOT of **TRUE\_\_**.
- ➔ **OR\_\_(B1, B2, B3)**: Logical OR operation between B1 and B2 based on B3.
- ➔ **NOR\_\_(B1, B2, B3)**: Logical NOR operation, the NOT of **OR\_\_**.
- ➔ **AND\_\_(B1, B2, B3)**: Logical AND operation between B1 and B2 based on B3.
- ➔ **NAND\_\_(B1, B2, B3)**: Logical NAND operation, the NOT of **AND\_\_**.
- ➔ **XOR\_\_(B1, B2, B3)**: Logical XOR operation, true if exactly one of B1 or B2 is true.
- ➔ **XNOR\_\_(B1, B2, B3)**: Logical XNOR operation, the NOT of **XOR\_\_**.
- ➔ **CMP\_\_(B1, B2, B3, B4)**: Compares B1 and B2 based on conditions B3 and B4.
- ➔ **NCMP\_\_(B1, B2, B3, B4)**: Logical NOT of **CMP\_\_**.
- ➔ **LOR\_\_(B1, B2, B3, M)**: Logical OR based on modes specified in M.
- ➔ **EQ\_\_(B1, B2, B3)**: Determines equality between B1 and B2 based on B3.
- ➔ **NEQ\_\_(B1, B2, B3)**: Determines inequality (NOT equal) between B1 and B2 based on B3.
- ➔ **IEQ\_\_(B1, B2, B3)**: Case-insensitive equality comparison between B1 and B2.



## ^ Boolean Operations

- ➔ **INEQ\_\_(B1, B2, B3)**: Logical NOT of **IEQ\_\_**.
- ➔ **GT\_\_(B1, B2, B3)**: Returns `true` if B1 is greater than B2.
- ➔ **LT\_\_(B1, B2, B3)**: Returns `true` if B1 is less than B2.
- ➔ **LE\_\_(B1, B2, B3)**: Returns `true` if B1 is less than or equal to B2.
- ➔ **GE\_\_(B1, B2, B3)**: Returns `true` if B1 is greater than or equal to B2.
- ➔ **IN\_\_(V, D, B)**: Determines if D is an element of array V and satisfies **TRUE\_\_**.
- ➔ **ORFT\_\_(B1, B2, B3)**: Returns true if B1 is false or B2 is true, based on B3.

### NOT\_\_(D)

```
1 function NOT__(D)
2 {
3     return ! D
4 }
```

function NOT<sub>(D)</sub>return!D

### NULL\_\_(D)

```
1 function NULL__(D)
2 {
3     return NOT__(D)
4 }
```

function NULL<sub>(D)</sub>returnNOT<sub>(D)</sub>

### FULL\_\_(D)

```
1 function FULL__(D)
2 {
3     return CMP__(D, "", "", 1)
4 }
```

function FULL<sub>(D)</sub>returnCMP<sub>(D,"","",1)</sub>

## TRUE\_\_(D, B)

```

1 function TRUE__(D, B)
2 {
3     if (B)
4         return FULL__(D)
5     else if (NOT__(NULL__(D)))
6         return 1
7     return 0
8 }

```

```
function TRUE(D,B)if(B)returnFULL(D)elseif(NOT(NULL(D)))return1return0
```

## IN\_\_(V, D, B)

```

1 function IN__(V, D, B)
2 {
3     return D in V && TRUE__(V[D], B)
4 }

```

```
function IN(V,D,B)returnDinVTRUE(V[D],B)
```

## FALSE\_\_(D, B)

```

1 function FALSE__(D, B)
2 {
3     return NOT__(TRUE\_\_(D, B))
4 }

```

```
function FALSE(D,B)returnNOT(TRUE_(D,B))
```

## OR\_\_(B1, B2, B3)

```

1 function OR__(B1, B2, B3)
2 {
3     return TRUE__(B1, B3) || TRUE\_\_(B2, B3)
4 }

```

```
function OR(B1,B2,B3)returnTRUE(B1,B3)||TRUE_(B2,B3)
```





### NOR\_\_(B1, B2, B3)

```
1 function NOR__(B1, B2, B3)
2 {
3     return NOT__(OR__(B1, B2, B3))
4 }
```

function NOR<sub>(B1,B2,B3)</sub>returnNOT<sub>(OR<sub>(B1,B2,B3)</sub>)</sub>

### ORFT\_\_(B1, B2, B3)

```
1 function ORFT__(B1, B2, B3)
2 {
3     # Return the result of OR__ with the negation of B1 and the truth value
↪ of B2
4     return OR__(FALSE__(B1, B3), TRUE__(B2, B3))
5 }
```

function ORFT<sub>(B1,B2,B3)</sub>ReturntheresultofOR<sub>w</sub>iththenegationofB1andthetruthvalueofB2returnOR<sub>(FALSE<sub>(B1,B3)</sub>,TRUE<sub>(B2,B3)</sub>)</sub>

### AND\_\_(B1, B2, B3)

```
1 function AND\_\_(B1, B2, B3)
2 {
3     return TRUE__(B1, B3) && TRUE__(B2, B3)
4 }
```

function AND\_\_(B1, B2, B3) return TRUE<sub>(B1,B3)</sub>TRUE<sub>(B2,B3)</sub>

### NAND\_\_(B1, B2, B3)

```
1 function NAND__(B1, B2, B3)
2 {
3     return NOT__(AND__(B1, B2, B3))
4 }
```

function NAND<sub>(B1,B2,B3)</sub>returnNOT<sub>(AND<sub>(B1,B2,B3)</sub>)</sub>

## XOR\_\_(B1, B2, B3)

```

1 function XOR__(B1, B2, B3)
2 {
3     # Return the result of OR__ with the combination of AND__ and AND__
4     return OR__(AND__(TRUE__(B1, B3), FALSE__(B2, B3)),
5                 AND__(FALSE__(B1, B3), TRUE__(B2, B3)))
6 }

```

function XOR<sub>(B1,B2,B3)</sub>ReturntheresultofOR<sub>w</sub>withthecombinationofAND<sub>a</sub>andAND<sub>r</sub>returnOR<sub>(</sub>AND<sub>(</sub>TRUE<sub>(</sub>B1,B3),FALSE<sub>(</sub>B2,B3)),AND<sub>(</sub>FALSE<sub>(</sub>B1,B3),TRUE<sub>(</sub>B2,B3))<sub>)</sub>

## XNOR\_\_(B1, B2, B3)

```

1 function XNOR__(B1, B2, B3)
2 {
3     return NOT__(XOR__(B1, B2, B3))
4 }

```

function XNOR<sub>(B1,B2,B3)</sub>returnNOT<sub>(</sub>XOR<sub>(</sub>B1,B2,B3)<sub>)</sub>

## CMP\_\_(B1, B2, B3, B4)

```

1 function CMP__(B1, B2, B3, B4)
2 {
3     # If B3 is true
4     if (B3) {
5         if (B4)
6             return B1 > B2
7         if (length(B4))
8             return B1 ~ B2
9         return B1 == B2
10    # Else if B3 has a length
11    } else if (length(B3)) {
12        if (B4)
13            return length(B1) > length(B2)
14        if (length(B4))
15            return length(B1) ~ length(B2)
16        return length(B1) == length(B2)
17    } else if (is_digit(B1, 1) && is_digit(B2, 1)) {
18        if (B4)
19            return +B1 > +B2
20        if (length(B4))
21            return +B1 ~ +B2
22        return +B1 == +B2
23    } else {
24        if (B4)
25            return "a" B1 > "a" B2

```



```

26         if (length(B4))
27             return "a" B1 ~ "a" B2
28         return "a" B1 == "a" B2
29     }
30 }

```

```
function CMP(B1,B2,B3,B4) if B3 is true if (B3) if (B4) return B1 > B2 if (length(B4)) return B1 B2 return B1 == B2 Else if B3 has a length the
```

### NCMP\_\_(B1, B2, B3, B4)

```

1 function NCMP__(B1, B2, B3, B4)
2 {
3     return NOT__(CMP__(B1, B2, B3, B4))
4 }

```

```
function NCMP(B1,B2,B3,B4) return NOT(CMP(B1,B2,B3,B4))
```

### LOR\_\_(B1, B2, B3, M)

```

1 function LOR__(B1, B2, B3, M, t)
2 {
3     # Determine mode based on M pattern: length or default
4     if (M ~ /^(l(e(n(g(t(h)?)?)?)?)?$/)) # Regex for 'length'
5         t = 0
6     else if (M ~ /^(d(e(f(a(u(l(t)?)?)?)?)?)?$/)) # Regex for 'default'
7         t = 1
8     # Full comparison based on t
9     if (FULL__(t)) {
10         if (B3)
11             return GT__(B1, B2, t) # Greater than comparison
12         else
13             return LT__(B1, B2, t) # Less than comparison
14     } else {
15         # Check if B1 and B2 are digits or M is 'string'
16         if (! (is_digit(B1, 1) && is_digit(B2, 1)) || M ~
17         ↪ /^(s(t(r(i(n(g)?)?)?)?)?$/))
18             t = "a" # Set t to 'a' for ASCII comparison
19         if (B3)
20             return GT__(t B1, t B2) # Concatenate t with B1 and B2,
21         ↪ compare
22         else
23             return LT__(t B1, t B2)
24     }
25 }

```

```
function LOR(B1,B2,B3,M,t) Determine mode based on M pattern: length or default if (M ~ /^(l(e(n(g(t(h)?)?)?)?)?$/)) # Regex for 'length' t = 0 else if (M ~ /^(d(e(f(a(u(l(t)?)?)?)?)?)?$/)) # Regex for 'default' t = 1 Full comparison based on t
```



if (FULL<sub>(t)</sub>) if (B3) return GT<sub>(B1,B2,t)</sub> Greater than comparison else return LT<sub>(B1,B2,t)</sub> Less than comparison else Check if B1 and B2 are digits or Mis' string' if  
 t = "a" Set t to 'a' for ASCII comparison if (B3) return GT<sub>(tB1,tB2)</sub> Concatenate t with B1 and B2, compare else return LT<sub>(tB1,tB2)</sub>

### EQ\_\_(B1, B2, B3)

```

1 function EQ__(B1, B2, B3)
2 {
3     return CMP__(B1, B2, B3)
4 }
```

function EQ<sub>(B1,B2,B3)</sub> return CMP<sub>(B1,B2,B3)</sub>

### NEQ\_\_(B1, B2, B3)

```

1 function NEQ__(B1, B2, B3)
2 {
3     return NCMP__(B1, B2, B3)
4 }
```

function NEQ<sub>(B1,B2,B3)</sub> return NCMP<sub>(B1,B2,B3)</sub>



### 3 Math

### 4 Strings

### 5 Type Validation

#### Type Validation

The following functions provide utilities to classify and manipulate numeric inputs in various formats. The examples demonstrate how to use these functions in practice.

- ➔ **\_\_is\_signed(N)**: Checks if the input number N has a + or - prefix.
- ➔ **\_\_get\_sign(N)**: Retrieves the sign (+ or -) of N if it is signed.
- ➔ **is\_integral(N, B)**: Verifies if N is an integer. The parameter B specifies whether to allow a sign prefix.
- ➔ **is\_signed\_integral(N)**: Checks if N is a signed integer.
- ➔ **is\_float(N, B)**: Determines if N is a floating-point number, with B controlling the allowance of a sign.
- ➔ **is\_signed\_float(N)**: Validates whether N is a signed floating-point number.
- ➔ **is\_digit(N, B)**: Checks if N is any numeric value (integer or float).
- ➔ **is\_signed\_digit(N)**: Checks if N is a signed numeric value (integer or float).

#### \_\_is\_signed(N)

```
1 function __is_signed(N)
2 {
3     return N ~ /^[ -][+]/
4 }
```

```
function is_signed(N) return N ~ /^[ -][+]/
```

#### \_\_get\_sign(N)

```
1 function __get_sign(N)
2 {
```

```

3     if (__is_signed(N)) {
4         return substr(N, 1, 1)
5     }
6

```

---

```
function getsign(N) if(is_signed(N)) return substr(N,1,1)
```

### is\_integral(N)

```

1 function is_integral(N, B, e)
2 {
3     if ((B && N ~ /^[(-|+)?[0-9]+$/ ) || (! B && N ~ /^[0-9]+$/))
4         e = 1
5     return e
6

```

---

```
function is_integral(N, B, e) if((BN /^[(-|+)?[0-9]+$/ ) || (! B N /^[0-9]+$/)) e = 1 return e
```

### is\_signed\_integral(N)

```

1 function is_signed_integral(N, e)
2 {
3     if (__is_signed(N) && is_integral(N, 1))
4         e = 1
5     return e
6

```

---

```
function is_signed_integral(N, e) if(is_signed(N) && is_integral(N,1)) e=1 return e
```

### is\_float(N, B)

```

1 function is_float(N, B, e)
2 {
3     if ((B && N ~ /^[(-|+)?[0-9]+[.][0-9]+$/ ) || (! B && N ~
↪ /^[0-9]+[.][0-9]+$/))
4         e = 1
5     return e
6

```

---

```
function is_float(N, B, e) if((BN /^[(-|+)?[0-9]+[.][0-9]+$/ ) || (! B N /^[0-9]+[.][0-9]+$/)) e = 1 return e
```



### is\_signed\_float(N)

```
1 function is_signed_float(N, e)
2 {
3     if (__is_signed(N) && is_float(N, 1))
4         e = 1
5     return e
6 }
```

*function is<sub>s</sub>igned<sub>f</sub>loat(N, e) if (<sub>i</sub>s<sub>s</sub>igned(N) is<sub>f</sub>loat(N, 1)) e = 1 return e*

### is\_digit(N, B)

```
1 function is_digit(N, B, e)
2 {
3     if (is_integral(N, B) || is_float(N, B))
4         e = 1
5     return e
6 }
```

*function is<sub>d</sub>igit(N, B, e) if (is<sub>i</sub>ntegral(N, B) || is<sub>f</sub>loat(N, B)) e = 1 return e*

### is\_signed\_digit(N)

```
1 function is_signed_digit(N, e)
2 {
3     if (__is_signed(N) && is_digit(N, 1))
4         e = 1
5     return e
6 }
```

*function is<sub>s</sub>igned<sub>d</sub>igit(N, e) if (<sub>i</sub>s<sub>s</sub>igned(N) is<sub>d</sub>igit(N, 1)) e = 1 return e*

## 6 Numbers Base

## 7 Internet

## 8 Miscellaneous

## 9 Standard Output

## 10 Structures