AWK - OPERATORS

Like other programming languages AWK also provide large set of operators. This tutorial explain AWK operators with suitable examples:

Arithmetic Operators

AWK supports following arithmetic operators:

Addition

It is represented by **plus** + symbol which adds two or more numbers. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { a = 50; b = 20; print "(a + b) = ", (a + b) }'
```

On executing the above code, you get the following result:

```
(a + b) =  70
```

Subtraction

It is represented by **minus** − symbol which subtracts two or more numbers. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { a = 50; b = 20; print "(a - b) = ", (a - b) }'
```

On executing the above code, you get the following result:

```
(a - b) =  30
```

Multiplication

It is represented by **asterisk** ∗ symbol which multiplies two or more numbers. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { a = 50; b = 20; print "(a * b) = ", (a * b) }'
```

On executing the above code, you get the following result:

```
(a * b) =  1000
```

Division

It is represented by **slash**/ symbol which divides two or more numbers. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { a = 50; b = 20; print "(a / b) = ", (a / b) }'
```

On executing the above code, you get the following result:

```
(a / b) =  2.5
```

Module

It is represented by **percent** symbol which finds module division two or more numbers. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { a = 50; b = 20; print "(a % b) = ", (a % b) }'
```

On executing the above code, you get the following result:

```
(a % b) =  10
```

Increment and Decrement Operators

AWK supports following increment and decrement operators:

Pre-increment

It is represented by **++**. It increments value of operand by **1**. This operator first increments value of operand then returns incremented value. For instance in below example this operator sets value of both operands, a and b to 11.

```
awk 'BEGIN { a = 10; b = ++a; printf "a = %d, b = %d\n", a, b }'
```

On executing the above code, you get the following result:

```
a = 11, b = 11
```

Pre-decrement

It is represented by **--**. It decrements value of operand by **1**. This operator first decrements value of operand then returns decremented value. For instance in below example this operator sets value of both operands, a and b to 9.

```
[jerry]$ awk 'BEGIN { a = 10; b = --a; printf "a = %d, b = %d\n", a, b }'
```

On executing the above code, you get the following result:

```
a = 9, b = 9
```

Post-increment

It is represented by **++**. It increments value of operand by **1**. This operator first returns the value of operand then it increments its value. For instance below example sets value of operand a to 11 and b to 10.

```
[jerry]$ awk 'BEGIN { a = 10; b = a++; printf "a = %d, b = %d\n", a, b }'
```

On executing the above code, you get the following result:

```
a = 11, b = 10
```

Post-decrement

It is represented by **--**. It decrements value of operand by **1**. This operator first returns the value of operand then it decrements its value. For instance below example sets value of operand a to 9 and b to 10.

```
[jerry]$ awk 'BEGIN { a = 10; b = a--; printf "a = %d, b = %d\n", a, b }'
```

On executing the above code, you get the following result:

```
a = 9, b = 10
```

Assignment Operators

AWK supports following assignment operators:

Simple assignment

It is represented by **=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { name = "Jerry"; print "My name is", name }'
```

On executing the above code, you get the following result:

```
My name is Jerry
```

Shorthand addition

It is represented by **+=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=10; cnt += 10; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 20
```

In above example first statement assigns value 10 to the variable **cnt** and in next statement shorthand operator increments its value by 10.

Shorthand subtraction

It is represented by **-=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=100; cnt -= 10; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 90
```

In above example first statement assigns value 100 to the variable **cnt** and in next statement shorthand operator decrements its value by 10.

Shorthand multiplication

It is represented by ***=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=10; cnt *= 10; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 100
```

In above example first statement assigns value 10 to the variable **cnt** and in next statement shorthand operator multiplies its value by 10.

Shorthand division

It is represented by **/=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=100; cnt /= 5; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 20
```

In above example first statement assigns value 100 to the variable **cnt** and in next statement shorthand operator divides it by 5.

Shorthand modulo

It is represented by **%=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=100; cnt %= 8; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 4
```

Shorthand exponential

It is represented by **^=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=2; cnt ^= 4; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 16
```

Above example raises value of **cnt** by 4.

Shorthand exponential

It is represented by ****=**. Below example illustrates this:

```
[jerry]$ awk 'BEGIN { cnt=2; cnt **= 4; print "Counter =", cnt }'
```

On executing the above code, you get the following result:

```
Counter = 16
```

This example also raises value of **cnt** by 4.

Relational Operators

AWK supports following relational operators:

Equal to

It is represented by **==**. It returns true if both operands are equal otherwise it returns false. Below example illustrates this:

```
awk 'BEGIN { a = 10; b = 10; if (a == b) print "a == b" }'
```

On executing the above code, you get the following result:

```
a == b
```

Not equal to

It is represented by **!=**. It returns true if both operands are unequal otherwise it returns false.

```
[jerry]$ awk 'BEGIN { a = 10; b = 20; if (a != b) print "a != b" }'
```

On executing the above code, you get the following result:

```
a != b
```

Less than

It is represented by **<**. It returns true if left side operand is less than right side operand otherwise it returns false.

```
[jerry]$ awk 'BEGIN { a = 10; b = 20; if (a < b) print "a < b" }'
```

On executing the above code, you get the following result:

```
a < b
```

Less than or equal to

It is represented by **<=**. It returns true if left side operand is less than or equal to right side operand otherwise it returns false.

```
[jerry]$ awk 'BEGIN { a = 10; b = 10; if (a <= b) print "a <= b" }'
```

On executing the above code, you get the following result:

```
a <= b
```

Greater than

It is represented by **>**. It returns true if left side operand is greater than right side operand otherwise it returns false.

```
[jerry]$ awk 'BEGIN { a = 10; b = 20; if (b > a ) print "b > a" }'
```

On executing the above code, you get the following result:

```
b > a
```

Greater than or equal to

It is represented by **>=**. It returns true if left side operand is greater than or equal to right side operand otherwise it returns false.

```
[jerry]$ awk 'BEGIN { a = 10; b = 10; if (a >= b) print "a >= b" }'
```

On executing the above code, you get the following result:

```
b >= a
```

Logical Operators

AWK supports following logical operators:

Logical AND

It is represented by **&&**. Below is the syntax of Logical AND operator.

```
expr1 && expr2
```

It evaluates to true if both expr1 and expr2 evaluate to true, otherwise it evaluates to false. expr2 is evaluated if and only if expr1 evaluates to true. For instance below example checks whether given single digit number is in octal format or not.

```
[jerry]$ awk 'BEGIN {num = 5; if (num >= 0 && num <= 7) printf "%d is in octal
format\n", num }'
```

On executing the above code, you get the following result:

```
5 is in octal format
```

Logical OR

It is represented by **||**. Below is the syntax of Logical OR operator.

```
expr1 || expr2
```

It evaluates to true if either expr1 or expr2 evaluates to true, otherwise it evaluates to false. expr2 is evaluated if and only if expr1 evaluates to false. Below simple example illustrates this.

```
[jerry]$ awk 'BEGIN {ch = "\n"; if (ch == " " || ch == "\t" || ch == "\n") print
"Current character is whitespace." }'
```

On executing the above code, you get the following result:

```
Current character is whitespace.
```

Logical NOT

It is represented by **exclamation mark!**. Below is the syntax of the same.

```
! expr1
```

It returns the logical compliment of expr1. If expr1 evaluates to true, it it returns 0; otherwise it returns 1. For instance below example checks whether string is empty or not.

```
[jerry]$ awk 'BEGIN { name = ""; if (! length(name)) print "name is empty string." }'
```

On executing the above code, you get the following result:

```
name is empty string.
```

Ternary Operator

We can easily implement condition expression using ternary operator. Below is the syntax of the same:

```
condition expression ? statement1 : statement2
```

When condition expression returns true, statement1 gets executed otherwise statement2 gets executed. For instance below example finds maximum number.

```
[jerry]$ awk 'BEGIN { a = 10; b = 20; (a > b) ? max = a : max = b; print "Max =", max}'
```

On executing the above code, you get the following result:

```
Max = 20
```

Unary Operators

AWK supports following unary operators:

Unary plus

It is represented by **+**. It multiplies single operand by **+1**.

```
[jerry]$ awk 'BEGIN { a = -10; a = +a; print "a =", a }'
```

On executing the above code, you get the following result:

```
a = -10
```

Unary minus

It is represented by **-**. It multiplies single operand by **-1**.

```
[jerry]$ awk 'BEGIN { a = -10; a = -a; print "a =", a }'
```

On executing the above code, you get the following result:

```
a = 10
```

Exponential Operators

This tutorial explain the two forms of exponential operators with suitable examples:

Exponential

It is an exponential operator which raises value of operand. For instance below example raises value of 10 by 2.

```
[jerry]$ awk 'BEGIN { a = 10; a = a ^ 2; print "a =", a }'
```

On executing the above code, you get the following result:

```
a = 100
```

Exponential

It is an exponential operator which raises value of operand. For instance below example raises value of 10 by 2.

```
[jerry]$ awk 'BEGIN { a = 10; a = a ** 2; print "a =", a }'
```

On executing the above code, you get the following result:

```
a = 100
```

String concatenation operator

Space is string concatenation operator which merge two strings. Below simple example illustrates this:

```
[jerry]$ awk 'BEGIN { str1="Hello, "; str2="World"; str3 = str1 str2; print str3 }'
```

On executing the above code, you get the following result:

```
Hello, World
```

Array membership operator

It is represented by **in**. It is used while accessing array elements. Below simple example prints array elements using this operator.

```
[jerry]$ awk 'BEGIN { arr[0] = 1; arr[1] = 2; arr[2] = 3; for (i in arr) printf "arr[%d]
= %d\n", i, arr[i] }'
```

On executing the above code, you get the following result:

```
arr[0] = 1
arr[1] = 2
arr[2] = 3
```

Regular Expression Operators

This tutorial explain the two forms of regular expressions operators with suitable examples:

Match

It is represented as **~**. It looks for a field that contains the match string. For instance below example prints lines which contains pattern **9**.

```
[jerry]$ awk '$0 ~ 9' marks.txt
```

On executing the above code, you get the following result:

```
2) Rahul Maths 90
5) Hari History 89
```

Not match

It is represented as **!~**. It looks for a field that does not contain the match string. For instance below example prints lines which does not contain pattern **9**.

```
[jerry]$ awk '$0 !~ 9' marks.txt
```

On executing the above code, you get the following result:

```
1) Amit Physics 80
3) Shyam Biology 87
4) Kedar English 85
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js