TATA CONSULTANCY SERVICES

Experience certainty.

Confidentiality Statement

Tata Code of Conduct

We, in our dealings, are self-regulated by a Code of Conduct as enshrined in the Tata Code of Conduct. We request your support in helping us adhere to the Code in letter and spirit. We request that any violation or potential violation of the Code by any person be promptly brought to the notice of the Local Ethics Counselor or the Principal Ethics Counselor or the CEO of TCS. All communication received in this regard will be treated and kept as confidential.

TATA CONSULTANCY SERVICES

Virtual ILP – Introduction to awk Filter

Content Manual Version 1.1

December 2014

(ILP Guwahati)

Table of Content

1. AWK inbuilt variables

AWK is supplied with good number of built-in variables which comes in handy when working with data files. We will see each AWK built-in variables with one or two examples to familiarize with them. Without these built-in variables it's very much difficult to write simple AWK code. These variable are used to format output of an AWK command, as input field separator and even we can store current input file name in them for using them within the script.

FS	field separator character (default blank & tab)
OFS	output field separator string (default blank)
RS	input record separator character (default newline)
ORS	output record separator string (default newline)
NF	number of fields in input record
NR	number of input record
FNR	output number of lines
FILENAME	name of current input file

Consider below db.txt as sample file.

~$ cat db.txt

John,29,MS,IBM,M,Married
Barbi,45,MD,JHH,F,Single
Mitch,33,BS,BofA,M,Single
Tim,39,Phd,DELL,M,Married
Lisa,22,BS,SmartDrive,F,Married

In order to make it simple we can divide above inbuilt variables in to groups on basis of their operations.

Group1: FS(input field separator), OFS
Group2: RS(Row separator) and ORS(Output record separator)
Group3: NR, NF and FNR
Group4: FILENAME variable

1.1. FS (Input Field Separator)

This variable is useful in storing the input field separator. By default AWK can understand only spaces, tabs as input and output separators. But if your file contains some other character as separator other than these mention one's, AWK cannot understand them.

For example UNIX password file which contain '**:**' as a separator. So in order to mention the input filed separator we use this inbuilt variable. We will see what issue we face if we don't mention the field separator for our db.txt.

Example: without using FS

Print first column data from db.txt file.

~$ awk '{print $1}' db.txt

Output:

John,29,MS,IBM,M,Married
Barbi,45,MD,JHH,F,Single
Mitch,33,BS,BofA,M,Single
Tim,39,Phd,DELL,M,Married
Lisa,22,BS,SmartDrive,F,Married

If you see entire file is displayed which indicates AWK do not understand db.txt file separator ",". We have to tell AWK what is the field separator.

Example: Using FS

List only first column data from db.txt file which have field separator as ','.

~$ awk 'BEGIN{FS=","}{print $1}' db.txt

Output:

John
Barbi
Mitch
Tim
Lisa

1.2. OFS (Output Field Separator)

This variable is useful for mentioning what is your output field separator which separates output data.

Example:

Display only 1st and 4th column and the separator between at output for these columns should be $.

```
~$ awk 'BEGIN{FS=",";OFS=" $ "}{print $1,$4}' db.txt
```

Output:

John $ IBM
Barbi $ JHH
Mitch $ BofA
Tim $ DELL
Lisa $ SmartDrive

Note: Space is give before and after $ in OFS variable to show better output.

1.3. RS (Row separator)

Row Separator is helpful in defining separator between rows in a file. By default AWK takes row separator as new line. We can change this by using RS built-in variable.

Example:

Convert a sentence to a word per line. We can use RS variable for doing it.

```
~$ echo "This is how it works" | awk 'BEGIN{RS=" "}{print $0}'
```

Output:

This
is
how
it
works

1.4. ORS (Output Record Separator)

This variable is useful for defining the record separator for the AWK command output. By default ORS is set to new line.

Example:

Print all the company names in single line which are in 4th column.

~$ awk -F',' 'BEGIN{ORS=" "}{print $4}' db.txt

Output:

IBM JHH BofA DELL SmartDrive

1.5. NF

This variable keeps information about total fields in a given row. The final value of a row can be represented with $NF.

Example: Consider abc.txt which contains below data:

Jones 2143 78 84 77
Gondrol 2321 56 58 45
RinRao 2122234 38 37
Edwin 253734 87 97 95
Dayan 24155 30 47

Print number of fields in each row in abc.txt.

~$ awk '{print NF}' abc.txt

Output:

5
5
4
5
4

Example:

Print last field in each row of abc.txt file.

~$ awk '{print $NF}' abc.txt

Output:

77
45
37
95
47

Note: If you observe above two examples we used Just NF for giving us the count of fields in a given row and $NF for displaying last element in each row. $NF will come handy when you are not sure what your last column number is.

1.6. NR

This variable keeps the value of present line number. This will come handy when you want to print line numbers in a file.

Example:

Print line number for each line in a given file.

~$ awk '{print NR, $0}' abc.txt

Output:

1 Jones 2143 78 84 77
2 Gondrol 2321 56 58 45
3 RinRao 2122234 38 37
4 Edwin 253734 87 97 95
5 Dayan 24155 30 47

This can be treated as cat command -n option for displaying line number for a file.

1.7. FNR

This variable keeps count of number of lines present in a given file/data. This will come handy when you want to print no of line present in a given file. This command is equivalent to wc -l command.

Example:

Print total number of lines in a given file.

```
~$ awk 'END{print FNR}' abc.txt
```

Output:

5

1.8. FILENAME

This variable contain file awk command is processing.

Example:

Print filename for each line in a given file.

```
~$ awk '{print FILENAME, NR, $0}' abc.txt
```

Output:

abc.txt 1 Jones 2143 78 84 77
abc.txt 2 Gondrol 2321 56 58 45
abc.txt 3 RinRao 2122234 38 37
abc.txt 4 Edwin 253734 87 97 95
abc.txt 5 Dayan 24155 30 47

2. Awk Built in Function

A function is a self-contained computation that accepts a number of arguments as input and returns some value. Awk has a number of built-in functions in two groups: arithmetic and string functions.

2.1. Arithmetic Functions

Nine of the built-in functions can be classified as arithmetic functions. Most of them take a numeric argument and return a numeric value. Below table summarizes these arithmetic functions with some examples.

Awk Function	Description
cos (***x***)	Returns cosine of ***x*** (***x*** is in radians).
exp (x)	Returns e to the power x.
index (s1,s2)	Position of string s2 in s1; returns 0 if not present
int (***x***)	Returns truncated value of ***x***.
log (***x***)	Returns natural logarithm (base- ***e***) of ***x***.
sin (***x***)	Returns sine of ***x*** (***x*** is in radians).
sqrt (***x***)	Returns square root of ***x***.
atan2 (***y*** , ***x***)	Returns arctangent of ***y*** / ***x*** in the range $-\pi$ to π.
rand ()	Returns pseudo-random number ***r***, where 0 <= ***r*** < 1.
srand (***x***)	Establishes new seed for rand(). If no seed is specified, uses time of day. Returns the old seed.

Examples:

sqrt(expr)

```
~$ awk 'BEGIN{
print sqrt(16);
print sqrt(0);
print sqrt(-12);
}'
```

Output:

4

0

nan

Here nan stands for not a valid number.

sin(expr)

```
~$ awk 'BEGIN {
print sin(90);
print sin(45);
}'
```

Output:

0.893997
0.850904

index()

```
~$ awk 'BEGIN { print  Index("peanut", "an") }'
```

Output:

3

substr(s,m,n)

```
~$ awk '{ print  substr(“deepak”,2,3) }'
```

Output

eep

rand()

```
~$cat rand.awk
BEGIN {
while(i<1000)
{
        n = int(rand()*100);
        rnd[n]++;
        i++;
}
for(i=0;i<=100;i++) {
        print i,"Occurred", rnd[i],     "times";
```

```
}
}
```

```
~$awk -f rand.awk
```

Output:

```
0 Occurred 6 times
1 Occurred 16 times
2 Occurred 12 times
3 Occurred 6 times
4 Occurred 13 times
.
.
.
```

2.2. String Functions

The built-in string functions are much more significant and interesting than the numeric functions. Because awk is essentially designed as a string-processing language, a lot of its power derives from these functions. Below table lists the string functions found in awk.

Awk's Built-In String Functions

Awk Function	Description
`gsub` (*r* , *s* , *t*)	Globally substitute's *s* for each match of the regular expression *r* in the string *t*. Returns the number of substitutions. If *t* is not supplied, defaults to **$0**.
`index` (*s* , *t*)	Returns position of substring *t* in string *s* or zero if not present.
`length` (*s*)	Returns length of string *s* or length of **$0** if no string is supplied.
`match` (*s* , *r*)	Returns either the position in *s* where the regular expression *r* begins or 0 if no occurrences are found. Sets the values of **RSTART** and **RLENGTH**.
`split` (*s* , *a* , *sep*)	Parses string *s* into elements of array *a* using field separator *sep* ; returns

Awk Function	Description
	number of elements. If *sep* is not supplied, **FS** is used. Array splitting works the same way as field splitting.
	Uses `printf` format specification for **expr**.
`sprintf` (" *fmt* ", *expr*)	
`sub` (*r* , *s* , *t*)	Substitutes *s* for first match of the regular expression *r* in the string *t*. Returns 1 if successful; 0 otherwise. If *t* is not supplied, defaults to **$0**.
`substr` (*s* , *p* , *n*)	Returns substring of string *s* at beginning position *p* up to a maximum length of *n*. If *n* is not supplied, the rest of the string from *p* is used.
`tolower` (*s*)	Translates all uppercase characters in string *s* to lowercase and returns the new string.
`toupper` (*s*)	Translates all lowercase characters in string *s* to uppercase and returns the new string.