

Gawk Begin Day 3

Dirk Janssen

August 2001

Topics, day 3

- Arithmetic operators, random numbers, string functions
 - substitution with `gsub` and `gensub`
 - regexp with `match`
 - `for`, `while`
 - arrays, array looping
-

Arithmetic operators and functions

`x%y` the remainder after dividing, $16\%3 \rightarrow 1$

`x^y` exponentiation, $2^4 \rightarrow 16$.

`int(x)` the integer part of a number, truncate the number towards 0 when `number > 0`.
`int(3.14159271) → 3`.

`sqrt(x)` the square root of a number.

`rand()` a random number between 0 and 1, more precisely $[0;1[$. NOTE no argument necessary or allowed.

`srand(x)` restart the random generator with *seed* x. Use as in `srand()` to restart with current time, for best results.

random number considerations

Caution: In most 'awk' implementations, including 'gawk', 'rand' starts generating numbers from the same starting number, or "seed", each time you run 'awk'. Thus, a program will generate the same results each time you run it. The numbers are random within one 'awk' run, but predictable from run to run. This is convenient for debugging, but if you want a program to do different things each time it is used, you must change the seed to a value that will be different in each run. To do this, use 'srand()'.

String considerations

Everything within quotation marks is literal, **except** a backslash (`\`). Backslash is the so-called escape character, used to enter characters that are otherwise difficult to type.

Most important ones are: `\t` for a tab character, `\n` for a newline (return) character, `\r` for the mac return character (dos return = `\r\n`), `\\` for a literal backslash.

There are differences between strings ("aa") and regexps (/aa/). These are subtle and hard to remember. Simply use `/aa/` for regexps and `"aa"` for strings and you will usually be ok. Occasionally, it comes in handy to use real strings (and esp. concatenation) to build so-called dynamic regexps (eg. `$0 ~ ("list" subjectnumber)`). Take some time to experiment when you want to do this.

String functions

index(in,what) Return first position of *what* in *in*. `index("this is a sentence", "is")`
→ 3.

length(x) The length in characters of *x*.

substr(source,from,length) That part of source that starts at the *from*th character and is *length* characters long. `substr("abacradabra", 4, 2)` → "cr"

substr(source,from) That part of source that starts at the *from*th character and until the end `substr("abacradabra", 8)` → "abra"

Search – Replace

Often you will want to search and replace in lines or fields. This can be done with `gsub` and `gensub`. I prefer the latter.

gensub(old,new,how,var) In *var*, change *old* into *new* and return the result. The variable itself is unchanged. Normally, use "g" for *how* to change globally, or 1 to change only the first occurrence. Gawk will search for *old*, a regexp (use `/. ./`), and replace it by *new*, a string (use `". ."`). If *var* is omitted, `$0` is used.

gsub(old,new,var) In *var*, change *old* into *new* everywhere. Return number of substitutions made. This function changes the value of your variable!

Normally, you will want to use `gensub` because it is easier to use. `gsub` comes in handy when you want to know the number of replacements made. I prefer `var=gensub(/ae/, "", "g", var)` over `gsub(/ae/, "", var)` because it is clear that *var* is changed.

gsub and CELEX

`gsub` can be used to change the backslashes in a CELEX file into spaces (actually not recommended because of words that contain spaces and because of empty fields).

```
{ gsub("\\", " ")
  print $0 }
```

```
1099\aarbbaan\39\C\aarde+baan\\'ard-ban\[VCC][CVVC]\N
1099 aarbbaan 39 C aarde+baan 'ard-ban [VCC][CVVC] N
```

For normal CELEX files, specify either `BEGIN{FS="\"}` (in a program) or the option `-F\\` (command line). The rabbit-like multiplication of backslashes (4 means 1) is due to the unix shell, similar to what is explained above: Shell maps `\\` to `\`, gawk does the same. Within double quotes, shell leaves the backslashes alone and these two commands are identical:

```
gawk -F\\ ' {print $7}' $dpl
gawk 'BEGIN{FS="\"}{print $7}' $dpl
```

regexp matching with match

The command `match(string, regexp)` will try to match string against regexp. It returns zero on failure. If successful, it sets `RSTART` to the index of the start of the match, and `RLENGTH` to the length of the match. This is precisely what `substr` needs to compute the part of the string that matched. This can come in very handy when you have troubles understanding regexps.

```
5126\angstbeeld\8\C\angst+beeld\\'ANzd-belt\[VCC][CVVC]\N
8950\beeldmooi\5\C\beeld+mooi\\'beld-moj\[VCC][CVVC]\A
```

```
BEGIN {
  FS="\"; OFS="\"
  match($7, / bel[dt]/ ) {
    print $2, substr($7, RSTART, RLENGTH) }
```

For statement

```
for ( INITIAL-ASSIGNMENT ; TEST ; STEP-EXPRESSION ) {
  BODY }
```

```
{ for(i=1; i<=5; i=i+1) {
  print $0 }
}
```

⇒ 1 2 3 4 5 (on separate lines)

Order of events: Assignment; Test; Body; Step; Test; Body; ... ; Step; Test. In this case:

```
i ← 1; i ≤ 5; print;
i ← 2; i ≤ 5; print;
i ← 3; i ≤ 5; print;
i ← 4; i ≤ 5; print;
i ← 5; i ≤ 5; print;
i ← 6; i > 5
```



```

$4=="Asia"  && $2 > largA { largA=$2 }
$4=="Europe" && $2 > largE { largE=$2 }
END { print largA, largE }

```

This is not ideal of course.

```

$4=="Asia" && $2 > largest["Asia"] {
  largest["Asia"]=$2 }
$4=="Europe" && $2 > largest["Europe"] {
  largest["Europe"]=$2 }
END {
  print largest["Asia"], largest["Europe"] }

```

Array looping

```

$2>largest[$4] {
  largest[$4]=$2 }
END {
  print largest["Asia"], largest["Europe"] }

```

If the number of continents is increased, the printing of all arrays subboxes becomes a nuisance. Hence, there is a special *for*-construct, which loops over all existing subboxes in an array:

```

$2>largest[$4] {
  largest[$4]=$2 }
END {
  for (idx in largest) { # DO NOT USE index
    print idx, largest[idx]}

```

The order in which indexes will appear is undetermined.

Array looping

```

BEGIN { a[1]=100
        a[2]=200
        a[7]=700
        for (i in a) { print "a[" _ i _ "]=" " _ a[i] }
        }
==>
a[7]= 700
a[1]= 100
a[2]= 200

```