awk – A UNIX tool to manipulate and generate formatted data

Alexander Voigt

Technische Universität Dresden
Institut für Kern- und Teilchenphysik

IKTP Computing Kaffee
10 January 2011

Content

1. History and Terminology
2. AWK – What is it made for?
3. Syntax
   - Calling awk from the command line
   - AWK program layout
   - The AWK view of the data stream
   - Variables and expressions
   - Conditionals
   - Loops
   - Functions
4. Examples
   - Manipulate program output
   - Execute commands from awk
   - Using awk together with Gnuplot
5. Implementations
6. Backup: Regular expressions

History and Terminology

AWK

- data driven programming language
- created 1977 at Bell Labs by Aho, Weinberger, Kernighan

awk

- refers to the UNIX program that runs programs written in the AWK programming language
- part of UNIX since Version 7 (1979)
- implementations: Bell Labs awk, GNU awk (gawk), tawk

AWK – What is it made for?

Intention:

- designed for processing text-based data, either in files or data streams
- benefits best realized when the data has some structure

AWK allows you to do:

- view text file/ stream as a database made up of records and fields
- use variables to manipulate the data
- use arithmetic and string operations (regular expressions, RE)
- use loops and conditionals
- define functions

Calling awk from the command line

Syntax to call awk:

```
$ awk [options] -f program-file text-file
$ awk [options] 'program-text' text-file
```

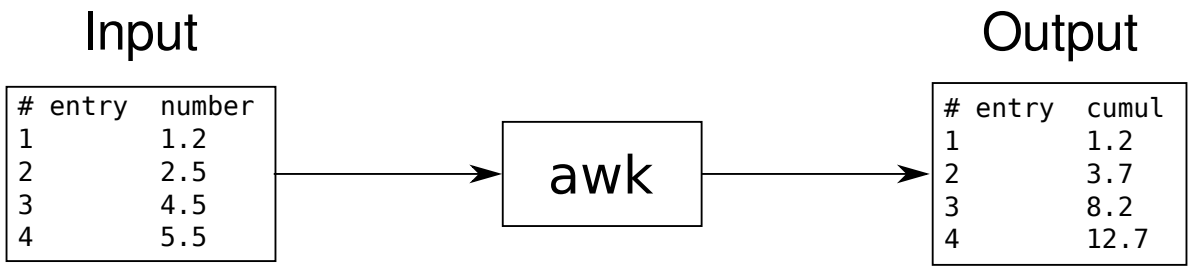
Examples using input from text file:

```
$ echo "This is some text" > text.txt
$ awk -f my-program.awk text.txt
$ awk '{ print "First field:" $1 }' text.txt
```

Example using input from stdin:

```
$ echo "This is some text" | awk '{ print "First field:" $1
  ↪  }'
This
```

AWK program layout

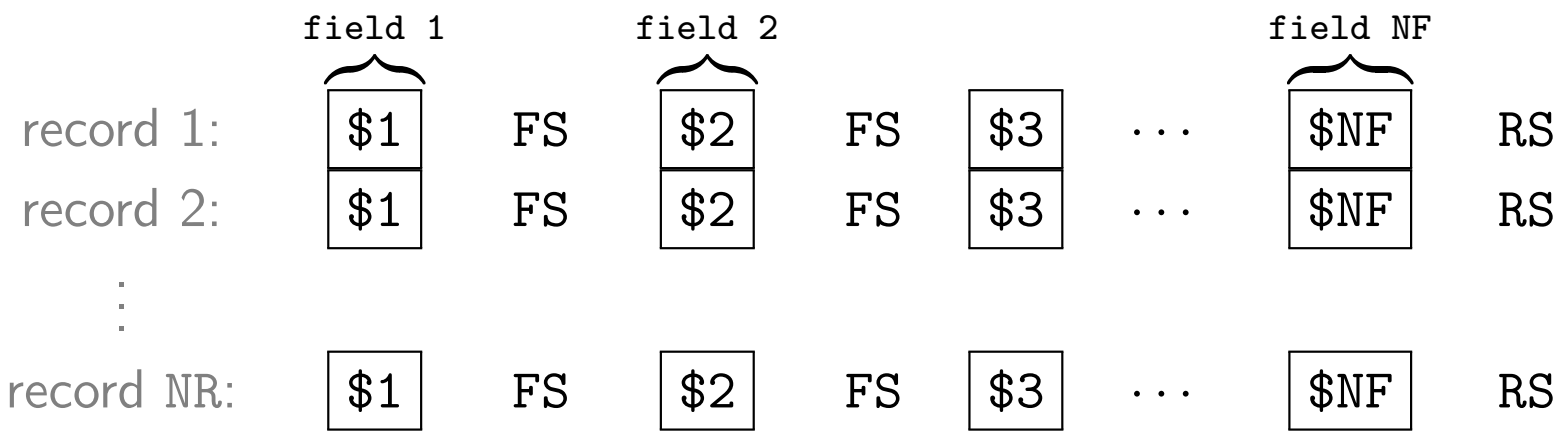


An AWK program consists of at least one of the following three parts:

```
1  BEGIN {
2    # executed once before any input is read
3  }
4
5  {
6    # main input loop: executed once for each input record
7  }
8
9  END {
10   # executed once after all input is read
11 }
```

The AWK view of the data stream

Decomposition of the input stream:



System variables:

- `FS` field separator
- `RS` record separator
- `NF` number of fields in the current record
- `NR` number of current input record

The AWK view of the data stream – Example 1

Example 1: Print all fields and separate them by "|"

```
$ cat addressbook1.txt
Richard Stallman, Musterstrasse 1, 12345 Musterhausen
Eben Moglen, Mustergasse 4, 23456 Musterdorf
Grady Booch, Examplestreet 5, 34567 Examplevillage

$ cat print-addressbook1.awk
BEGIN {
   FS=", "; RS="\n"
}
{
   print $1 "|" $2 "|" $3
}

$ awk -f print-addressbook1.awk addressbook1.txt
Richard Stallman|Musterstrasse 1|12345 Musterhausen
Eben Moglen|Mustergasse 4|23456 Musterdorf
Grady Booch|Examplestreet 5|34567 Examplevillage
```

The AWK view of the data stream – Example 2

Example 2: Using different field and record separators

```
$ cat addressbook2.txt
Richard Stallman
Musterstrasse 1
12345 Musterhausen

Eben Moglen
Mustergasse 4
23456 Musterdorf

Grady Booch
Examplestreet 5
34567 Examplevillage

$ cat print-addressbook2.awk
BEGIN { FS="\n"; RS="" }
{ print $1 "|" $2 "|" $3 }

$ awk -f print-addressbook2.awk addressbook2.txt
Richard Stallman|Musterstrasse 1|12345 Musterhausen
Eben Moglen|Mustergasse 4|23456 Musterdorf
Grady Booch|Examplestreet 5|34567 Examplevillage
```

Variables and expressions

Variables:

- no declaration necessary
- have a string value and a numeric value; used value depends on context (Strings that are not convertible into a number have a numeric value 0)
- no initialization necessary; automatically initialized to the empty string, which acts like 0 if used as a number

Examples:

```
1 x = 1         # string value = "1",  numeric value = 1
2 x = "a"       # string value = "a",  numeric value = 0
3 x = "a" "b" # string value = "ab", numeric value = 0
```

Arithmetic operators: `+ - * / % ^ ++ -- += -= *= /= %= ^=`

Examples:

```
1 x = 1
2 y = 2
3 print x " / " y " = " x/y
```

Conditionals

Syntax:

```
1 if (expression) {
2   statement
3 }
4 [else {
5   statement
6 }]
```

Table: Rational operators

<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
!=	Not equal to
~	Matches regular expression
!~	Does not match reg. exp.

Table: Boolean operators

||	Logical OR
&&	Logical AND
!	Logical NOT

Conditionals – Example

Print filename if condition is "True" or "true" and number is even:

```
$ cat data1.txt
# filename        condition       number
file1.txt         True            1
file2.txt         true            2
file3.txt         false           3
file4.txt         False           4

$ cat print-data1.awk
{
    # skip record if first field starts with '#'
    if ($1 ~ /^#/) next;

    if ($2 ~ /(T|t)rue/ && $3 % 2 == 0) {
        print $1
    }
}

$ awk -f print-data1.awk data1.txt
file2.txt
```

Loops

While loop:

```
while (expression) {
  statement
}
```

Do loop:

```
do {
  statement
} while (expression)
```

For loop:

```
for (set_counter; test_counter; increment_counter) {
  statement
}
```

Example: print all fields of current record

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

Functions

Syntax:

```
function name (parameter_list) {
  statement
}
```

Example:

```
function MyFunc (X, Y,  Z) {
  # X, Y, Z are local variables
  W = 1  # define global variable
  print X, Y, Z, W
}
{
  # call the function; Z is initialized to the empty string
  MyFunc($1, "var2")
  # at this point W has the value 1
  print W
}
```

Remarks:

- function parameters are local variables
- variables defined in the function body are global

Example 1 – Manipulate program output

Get the current platform name:

```
1 $ uname -a
2 Linux alex-laptop 2.6.35-24-generic #42-Ubuntu SMP Thu Dec
    ↪  2 02:41:37 UTC 2010 x86_64 GNU/Linux
3
4 $ uname -a | awk '{ print $1 " operating system" }'
5 Linux operating system
```

Example 2 – Execute commands from awk

Kill LSF batch jobs by host name:

```
$ bjobs
JOBID      USER    STAT  QUEUE  EXEC_HOST   SUBMIT_TIME
115236337  avoigt  PEND  1nd    lxbsp414    Jan  6 19:58
115236338  avoigt  PEND  1nd    lxbsp414    Jan  6 19:58
115236339  avoigt  PEND  1nd    lxbsp415    Jan  6 19:58
115236340  avoigt  PEND  1nd    lxbsp415    Jan  6 19:58

$ bjobs | awk 'if ($5~/lxbsp414/) { cmd="bkill " $1;
  ↪  system(cmd); }'
Job <115236337> is being terminated
Job <115236338> is being terminated
```

Example 2 – Using awk together with Gnuplot

Plot first column vs. cumulative values of first column:

```
$ cat data2.txt
1
2
3
# ...
9
10
```

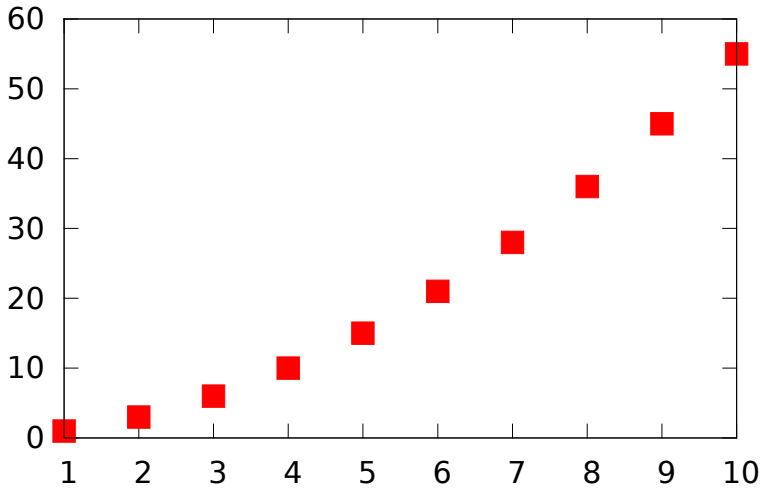
```
awk '{x=x+$1; print $1,x}' data2.txt
1 1
2 3
3 6
4 10
5 15
6 21
7 28
8 36
9 45
10 55
```

Example 2 – Using awk together with Gnuplot

Plot first column vs. cumulative values of first column:

```
awk '{x=x+$1; print $1,x}' data2.txt
1 1
2 3
3 6
# ...
9 45
10 55
```

```
plot "<awk '{x=x+$1; print $1,x}' data2.txt"
```



Implementations

Bell Labs awk descendant of V7 awk.

GNU awk [recommended!] more features: extended RE, external variables, additional functions (gensub(), systime(), strftime()), ...

tawk AWK compiler for DOS, Windows, Solaris. Many new features: array sorting, RE flags, more I/O functions, ...

Regular expressions

Regular expression = string which describes a set of strings by use of syntactic rules

meta symbol	description	examples
`[]`	match single character within [ ]	`[abc], [a-z]`
`( )`	marked subexpression	`(a|b)`
`{ }`	match between m and n times	`a{1,3}`
`|`	choice operator	`a|b`
`?`	match zero or one time	`a?`
`+`	match one or more times	`a+`
`*`	match zero or more times	`a*`
`-`	define range	`[a-zA-Z]`
`^`	match starting position in string	`^abc`
`$`	match ending position of string	`xyz$`
`.`	match any single character	`ab.de`
`\n`	*n*th marked subexpression	`(ab)\1`