

AWK

Marek Kozłowski



Faculty of Mathematics and Information Sciences
Warsaw University of Technology

- In short: *AWK = C for dummies + regular expressions*
- Analyzes text files (or text streams when used in pipelines).
- Text files are seen as databases. Records are identified with lines and fields – with blank separated words. Default record and field separators can be redefined (RS and FS variables).
- Number of words (fields) in lines (records) can be variable (NF variable).
- Files are processed line by line (record by record).

Running AWK Scripts

1. AWK script inline:

```
$ awk 'script_here' file.txt
```

2. AWK script body moved to a separate file:

```
$ awk -f script.awk file.txt
```

3. Standalone AWK script:

```
$ head -n1 script.awk
    #!/usr/bin/awk -f
$ chmod +x script.awk
$ ./script.awk file.txt
```

AWK Script Structure

```
#!/usr/bin/awk -f

BEGIN      {instructions_B}
condition_1 {instructions_1}
condition_2 {instructions_2}
condition_3 {instructions_3}
...
condition_n {instructions_n}
END        {instructions_E}
```

- BEGIN and END sections are optional.
- Empty condition is substituted with *true*.
- Omitted instruction block is substituted with *{print current line on stdout}*.

AWK Script Processing

```
if (present(BEGIN)) {instructions_B}  
  
while (read(line))  
    for (i=1; i<=n; i++)  
        if (condition_i(line))  
            instructions_i(line)  
  
if (present(END)) {instructions_E}
```

Special Variables

- For simplification we use the terms *word* and *line* instead of more general *field* and *record*.
- `$1`, `$2`, ..., `$(NF-1)`, `$NF` – 1^{st} , 2^{nd} , ..., last word on the current line.
- `$0` – the current line,
- `NF` – number of words on the current line,
- `NR` – number of lines processed till now,
- `FS`, `OFS` – input/output field separators (by default: any sequence of blanks),
- `RS`, `ORS` – input/output record separators (by default EOL),
- `ARGC`, `ARGV` – same as in C.

Instructions and Operators

- AWK instructions are the same as C ones and share the same syntax. The only C instruction not implemented in AWK is the `switch/case` one.
- AWK implements all C logical, arithmetical and comparison operators (including `?:`).
- Most C basic arithmetic functions are available and work the same way in AWK.
- AWK implements `printf()` which uses C syntax and format modifiers.
- AWK offers a slightly different set of (easy) string processing functions.

C for Dummies

- If there is no ambiguity parenthesis can be safely omitted.
- Semicolons are optional, required only for separating instructions in the same line.
- Variables are not declared/defined before the first use.
- Type casting is performed automatically based on a use context.
- There are no pointers.
- Empty operator concatenates strings.
- Strings can be compared using arithmetic operators (`==`, `!=`, `<`, `<=`, `>` and `>=`).
- If there is no need for special formatting then `print` – a simple alternative to `printf` can be used.
- Tables are associative; they act as dictionaries.
- Shell-like comment (`#`) can be used.

Regular Expression Matching Operator

Variable contains a substring described by a regexp

```
some_variable ~ /some_regexp/
```

Example: the following conditions are equivalent (thanks to automagical casting):

```
somevar>=1000 && somevar<2000  
somevar ~ /^1[0-9]{3}$/
```

Example: looks strange but it's perfectly OK

```
i=32; j=1 # numbers; we don't declare variables  
k=i j # 321 (string operator automagically casts to strings)  
k*=2 # 642 (numeric operator casts again to a numeric value)  
k ~ /[246]{3}/
```

Examples: AWK vs Shell Commands

```
cat somefile
```

```
$ awk '1' somefile
```

```
head somefile
```

```
$ awk 'NR<=10' somefile
```

```
wc -l somefile
```

```
$ awk 'END {print NR}' somefile
```

```
wc -w somefile
```

```
$ awk 'BEGIN {i=0} {i+=NF} END {print i}' somefile
```

```
grep -E 'somepattern' somefile
```

```
$ awk '$0~/somepattern/' somefile
```

Example: Associative Tables

sort

```
#!/usr/bin/awk -f

{tab[NR-1]=$0}

END {
for (i=0;i<NR;i++) # bubble sort
    for (j=0;j<NR;j++)
        if (tab[j+1]<tab[j])
            {temp=tab[j]; tab[j]=tab[j+1]; tab[j+1]=temp}
for (i=0;i<NR;i++) print tab[i]}
```

Example: Advanced Field Management

- Field separators can be redefined in the BEGIN section.
- Records can be modified in-place.

✎ Modify /etc/passwd: if a UID contains only odd numbers then increase GID by 1000

```
$ ./script.awk /etc/passwd > newpasswd
```

```
#!/usr/bin/awk -f
```

```
BEGIN {FS=":"; OFS=":"}
```

```
$3~/^[13579]+$/{ $4+=1000 } # modify record in-place  
1
```

Reading Script Parameters

```
$ ./script.awk somefile.txt one two three ...
```

```
#!/usr/bin/awk -f

# BEGIN block that parses parameters given to the script
# and removes them from the command line.
BEGIN {
  for (i=2; i<ARGC; i++)
    params[i-2] = ARGV[i]
  par_nr=ARGC-2
  ARGC=2}
```