NOTES ON AWK

Frédéric GALLIANO

Université Paris-Saclay, Université Paris Cité, CEA, CNRS, AIM, 91191, Gif-sur-Yvette, France

January 5, 2024

Contents

---

1 GENERAL SYNTAX

1.1 Call

Awk is a shell command. It can be executed on a file:

```
awk '<pattern1> { <action1> } ... <patternN> { <actionN> }' <optional-declarations> <file>
```

or with a redirection:

```
<unix-command> | awk '<pattern1> { <action1> } ... <patternN> { <actionN> }' <optional-decla
```

1.2 Data

Data can thus be either a text file, or the output of a shell command. These data are splitted in a series of records (by default, the lines of the file). The records are splitted in fields (by default, the words).

1.3 Actions

Actions are instructions to apply to data depending on the conditions given by `<pattern>`.

The default action is `print`, *i.e.* `<pattern>` ⇔ `<pattern> { print }`.

1.4 Conditions

Conditions are boolean instructions selecting the data on which the actions will apply. They can also be `BEGIN` or `END` instructions.

By default, all the data are considered.

1.5 Declarations

Optional variable declarations can be added at the end (*e.g.* change the value of `FS`, *etc.*).

---

2 VARIABLES AND FUNCTIONS

2.1 Identifiers

`$i` → identify the i^{th} field of the record ($1 \leq i \leq N$).
`$0` → identify the whole record.

2.2 Predefined Variables

`RS` (*Record Separator*) → delimiter to split the data into different records. By default, it is `\n`.
`ORS` (*Output Record Separator*) → similar to `RS`, but for outputs. By default, it is `\n`.
`NR` (*Number of Records*) → number of current lines, by default ($1 \leq NR \leq N$). It is 0 for an empty file.
`FS` (*Field Separator*) → delimiter to split the records into fields, during the reading. By default,it is `SPC`.
`OFS` (*Output Field Separator*) → similar to `FS`, but for outputs. By default, it is `SPC`.
`NF` (*Number of Fields*) → number of fields in the current record ($1 \leq NF \leq N$). It is 0 for a blank line.

2.3 Booleans

`0` → false.
`1 or non zero` → true.
`&&` → and.
`||` → or.
`!` → not.
`+<var>` → true if `<var>` is a numerical variable.

2.4 Predefined Functions

`print(<val1>, ... , <valN>)` → write `<val1>` to `<valN>`, which can be the value of a variable, a field or a string between quotes.
`printf("<format>",<var>)` → where the format can be a combination of strings and instructions:
- **`%d`** → integer, free format;
- **`%<n>d`** → integer, on `<n>` columns;
- **`%f`** → real, free format;
- **`%<n>.<d>f`** → real with `<d>` decimals, on `<n>` columns;
- **`%s`** → string, free format;
- **`%<n>s`** → string, on `<n>` columns.

`toupper(<var>)` → convert `<var>` in upper case.
`substr(<var>,<first-char>,<length>)` → extract from the string `<var>` the characters between `<first-char>` and `<first-char>+<length>`.
`split(<varIN>,<varOUT>,<sep>)` → split the string `<varIN>` in a table `<varOUT>` according to the separator `<sep>`.
`length(<var>)` → number of characters in `<var>`.

2.5 Operations

Simple operations can be done on variables. In particular:
Initialization → `<var>=<val>` (by default, a variable is initialized to 0);
Incrementation → `<var>++`;
Arbitrary Incrementation → `<var>+=<incr>`.

2.6 Tables

Tables are associative. They work as a `dict` in Python: `ARR[<var>]` where `<var>` is not necessarily an index, but can be a field.

---

3 ALGORITHMIC

3.1 Instructions at the Beginning or the End of a Read

`BEGIN { <action> }` → perform `<action>` before the reading. This can used to initialize variables.
`END { < action> }` → perform `<action>` after writing.

3.2 Structure

Iterative schemes → `{ for (<arr1> in <arr2>) <action> }`.
Decision schemes → `{ if (<cond1>) { <action1> } else if (<cond2>) { <action2> } else { <action3> }`.
