awk – An Introduction

17/10/2019

Software Systems Lab

Background

- AWK – A Programming Language
- The letters are initials of its creators – Alfred Aho, Peter Weinberger, Brian Kernighan
- Created in 1977 at Bell Labs for Unix-like systems
  - Revised and expanded during 1985-88, released as GNU AWK.
- awk followed sed (developed around 1973-74)
  - awk inspired **Perl**, another text-processing and reporting language

Trivia

source:https://en.wikipedia.org/wiki/Auk

- Auk is the name of this bird
- Pronounced oh-k

- Purpose - data extraction and reporting
  - Design goal - manipulate strings and numbers
  - Input – text files
  - Example:
    - Get the name of all airports within 500kms from Hubballi.

Input

```
HBX,Hubballi,0
IXG,Belagavi,74
TIR,Tirupati,660
HYD,Hyderabad,498
PNQ,Pune,431
BLR,Bengaluru,411
BOM,Mumbai,575
COK,Cochin,865
GOI,Dabolim,196
```

Output

From	To	Distance	Comments
HBX	IGX	74	Hubballi-Belagavi
HBX	HYD	498	Hubballi-Hyderabad
HBX	PNQ	431	Hubballi-Pune
HBX	BLR	411	Hubballi-Bengaluru
HBX	GOI	198	Hubballi-Dabolim

How does it work?

- Line-by-line processing
  - A file is a sequence of *records.*
    - By default, a line of an input text file is a record
  - A record is a sequence of *fields*
    - By default, a word in a line is a field: first word = field 1, second word = field 2, and so on..
- awk program - a sequence of pattern-action statements

```
condition { action }
condition { action }
        .
        .
condition { action }
```

awk Program

```
BEGIN { action_pre }

condition_1 { action_1 }
      .
      .
condition_n { action_n }

END { action_post }
```

Execute `action_pre` in BEGIN block
`while` (end of file is not reached) {
- process line `i`:
  - if line `i` has a pattern matching `condition_1`, execute `action_1`
    .
  - if line `i` has a pattern matching `condition_n`, execute `action_n`

}
Execute `action_post` in END block

= Optional Blocks

- *What happens if* `action_i` *is missing?*
  - *Whole line is displayed*

Language Features

- Number of built-in variables
  - current-line variables (`NF, NR, $0, $n`)
  - Others: `ARGC, ARGV, RS, FS, IGNORECASE, etc.`
- Powerful regular expression handling
- Operators manipulating expressions
  - `+, -, *, /, %, ++, --, ^, **, <, >, <=, >=, in, etc.`

All these features make AWK *very powerful*

Programming Examples