

# awk – An Introduction

17/10/2019

Software Systems Lab

# Background

- **AWK** – A Programming Language
- The letters are initials of its creators – Alfred **A**ho, Peter **W**einberger, Brian **K**ernighan
- 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 **P**erl, 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	Output			
HBX,Hubballi,0	From	To	Distance	Comments
IXG,Belagavi,74	HBX	IGX	74	Hubballi-Belagavi
TIR,Tirupati,660	HBX	HYD	498	Hubballi-Hyderabad
HYD,Hyderabad,498	HBX	PNQ	431	Hubballi-Pune
PNQ,Pune,431	HBX	BLR	411	Hubballi-Bengaluru
BLR,Bengaluru,411	HBX	GOI	198	Hubballi-Dabolim
BOM,Mumbai,575				
COK,Cochin,865				
GOI,Dabolim,196				

# 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: ARGV, ARGV, RS, FS, IGNORECASE, etc.
- Powerful regular expression handling
- Operators manipulating expressions
  - +, -, \*, /, %, ++, --, ^, \*\*, <, >, <=, >=, in, etc.

*All these features make AWK very powerful*

# Programming Examples



# Regular Expressions, Arrays, Control Structures, User-defined Functions

24/10/19

# Regular Expressions

- Notations for specifying and matching strings of characters
- You have seen them in *shell program*, MS-DOS, some text editors...
- In awk, they are used as string-matching patterns
  - Enclose the string in between forward-slashes - / /  
E.g. /Man/ {print \$0}
  - meaning: whenever a line of input **contains** the string specified in between / /, take some action.* Here /Man/ matches lines such as:
    - “Mangoes are tasty”
    - “Woman and Man are created equal”
    - “Man-pages in Linux are a very useful source of information”
- Operator ~ and !~

# Basic Regular Expressions

- Metacharacters: \, (, ), ., \*, +, ?, [, ], ^, \$, |
- Basic Regular Expression is one of the following:

Properties	Example
Non metacharacter	/A/ - matches 'A' only
Escape sequence	\t - matches a tab
Quoted metacharacter	\* - matches '*'
^	/^A/ - matches beginning of the string
\$	/\$A/ - matches end of string
.	./ matches any single character
Character class	[Aa] - matches 'a' or 'A'
Abbreviated character class	[A-Za-Z] - matches a single letter (case insensitive) [^0-9] - matches single non-digit character

# Exercise

- *What would `[+-]` match? how about `[-+]`?*
- *What about `^[^^]`?*
- Inside a character class `[]` only `\`, `^`, and `-` have special meanings. Further,
  - `^` only when placed at the beginning of `[]`
  - `-` between two characters
- *Show all hidden files in a directory (file names that begin with `'.'`)*

- *Range patterns: pat1, pat2*

*/NR=2/, /NR=3/*

*/Europe/, /Africa/*

# Complex Regular Expressions

- Combining Basic Regular Expressions

Properties	Example
Alternation	$A B$ – matches A or B
Concatenation	$AB$ – matches A followed by B
Closure	$A^*$ - matches zero or more occurrences of A
Positive closure	$A^+$ - matches one or more occurrences of A
Zero or one / optional	$A?$ – matches no character or exactly single A
Parenthesis	$(A B)(C D)$ – matches A or B followed by C or D

# Arrays

- Need not be explicitly declared
- Need not specify size either
- Example:

`Arr[NR]=$0`

- Array subscripts are strings
- Only 1D arrays supported