Network Programming

Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

AWK Utility

One of the most prominent text-processing utility on GNU/Linux

1. Overview

- AWK is an **interpreted programming language**

- It is very powerful and specially designed for **text processing**

- It is often referred to as **GNU AWK**

  The version of AWK that GNU/Linux distributes is written and maintained by the Free Software Foundation (FSF)

1. Overview

- Types of AWK:
  - **AWK**

    Original AWK from AT & T Laboratory
  - **NAWK**

    Newer and improved version of AWK from AT & T Laboratory
  - **GAWK**

    ✓ It is GNU AWK

    ✓ All GNU/Linux distributions ship GAWK

    ✓ It is fully compatible with AWK and NAWK

1. Overview

- Typical Uses of AWK:

  Myriad of tasks can be done with AWK:

  - Text processing
  - Define variables to store data
  - Use structured programming concepts to add logic to data processing

    if-then statements and loops
  - **Producing formatted text reports**

    Generate formatted reports by extracting data elements within the data file and repositioning them in another order or format
  - Performing arithmetic operations
  - Performing string operations, and many more

2. Environment

- AWK is available by default on most GNU/Linux distributions
- Can use **which** command to check whether it is present on your system or not
- Installation Using Package Manager

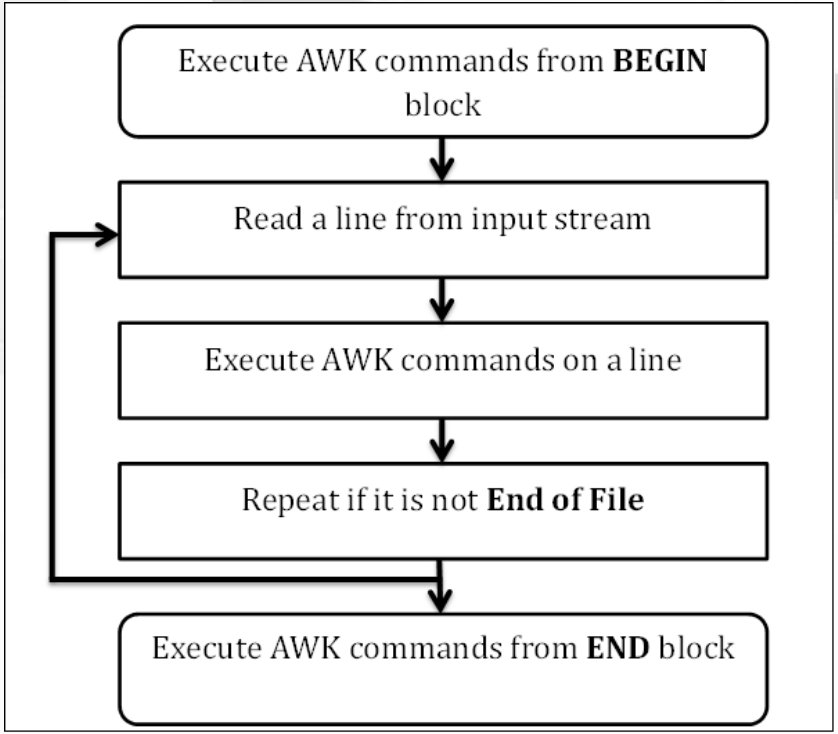
  Ubuntu: Advance Package Tool (APT) package manager

  - sudo apt-get update
  - sudo apt-get install gawk
- After installation, ensure that AWK is accessible via command line

  which awk

3. Workflow

AWK follows a simple workflow − Read, Execute, and Repeat



3. Workflow

- **Read**

  AWK reads a line from the input stream (file, pipe, or stdin) and stores it in memory

- **Execute**
  - All AWK commands are applied sequentially on the input
  - By default AWK execute commands on every line
  - We can restrict this by providing patterns

- **Repeat**

  This process repeats until the file reaches its end

4. Program Structure

- **BEGIN block**
  - Syntax: **BEGIN {awk-commands}**
  - The BEGIN block gets executed at program start-up
  - It executes only once
  - This is good place to initialize variables
  - BEGIN is an AWK keyword and hence it must be in **upper-case**
  - **Note:** This block is **optional**

4. Program Structure

- **Body Block**
  - Syntax: **/pattern/ {awk-commands}**
  - The body block applies AWK commands on **every input line**
  - By default, AWK executes commands on every line. We can restrict this by providing patterns
  - There are no keywords for the Body block

4. Program Structure

- **END Block**
  - Syntax: **END {awk-commands}**
  - The END block executes at the end of the program.
  - END is an AWK keyword and hence it must be in **upper-case**
  - **Note:** this block is **optional**

4. Program Structure

Example

- Create a **data file** (marks.txt)

```
1)  Amit    Physics  80
2)  Rahul   Maths    90
3)  Shyam   Biology  87
4)  Kedar   English  85
5)  Hari    History  89
```

- Write AWK code:

```
awk 'BEGIN { printf "Sr No\tName\tSub\tMarks\n“ }
{ print }' marks.txt
```

5. Basic Syntax

- AWK is simple to use
- We can provide AWK commands:
  - either **directly** from the **command line**
  - or in the form of a **text file** containing AWK commands
- **AWK Command Line**

  Specify an AWK command within **single quotes**
  - **awk [options] ‘awk commands’ [dataFile]**
  - Example: awk '{print}' marks.txt

5. Basic Syntax

- **AWK Program File**

  Provide AWK commands in a script file

  - **awk [options] -f awkCommandFile [dataFile]**
  - Example: awk -f command.awk marks.txt

5. Basic Syntax

- **AWK Standard Options**
  - **The -v option**
    - ✓ assigns a value to a variable
    - ✓ allows assignment before the program execution
    - ✓ Example: **awk -v name=Jerry 'BEGIN{printf "Name = %s\n", name}'**
  - **The --dump-variables[=file] option**
    - ✓ prints a sorted list of global variables and their final values to file
    - ✓ The default file is **awkvars.out**
    - ✓ Example:
      - ➢ awk --dump-variables ‘’
      - ➢ cat awkvars.out

5. Basic Syntax

- **AWK Standard Options**
  - The --help option
  - prints the help message on standard output
  - Example: **awk --help**

6. Basic Examples

Use the data file **marks.txt**

- **Printing Column or Field**
  - instruct AWK to print only certain columns from the input field
  - Example: **awk '{print $3 "\t" $4}' marks.txt**

  $3 and **$4** represent the third and the fourth fields respectively from the input record

- **Printing All Lines**
  - By default, AWK prints all the lines that match pattern
  - Example:
    - ✓ **awk '/a/ {print $0}' marks.txt**
    - ✓ **awk '/a/' marks.txt**

6. Basic Examples

- **Printing Columns by Pattern**
  - instruct AWK to print only certain fields
  - Example: **awk '/a/ {print $3 "\t" $4}' marks.txt**

- **Printing Column in Any Order**

  Example: **awk '/a/ {print $4 "\t" $3}' marks.txt**

6. Basic Examples

- **Counting and Printing Matched Pattern**
  - count and print the number of lines for which a pattern match succeeded
  - Example: **awk '/a/{++cnt} END {print "Count = ", cnt}' marks.txt**

- **Printing Lines with More than 18 Characters**
  - print only those lines that contain more than 18 characters
  - Example: **awk 'length($0) > 18' marks.txt**
    - ✓ AWK provides a built-in **length** function that returns the length of the string
    - ✓ $0 variable stores the entire line

7. Built-in Variables

Standard AWK variables

- **ARGC**
  - ✓ the number of arguments provided at the command line
  - ✓ Example: **awk 'BEGIN {print "Arguments =", ARGC}' One Two Three Four**
- **ARGV**
  - ✓ an array that stores the command-line arguments
  - ✓ Example:

```
awk 'BEGIN {
        for (i = 0; i < ARGC; ++i) {
                printf "ARGV[%d] = %s\n", i, ARGV[i]
        }
}' one two three four
```

7. Built-in Variables

Standard AWK variables

- **FILENAME**
  - ✓ represents the current file name
  - ✓ **Note:** FILENAME is undefined in the BEGIN block
  - ✓ Example: **awk 'END {print FILENAME}' marks.txt**
- **NF**
  - ✓ represents the number of fields in the current record
  - ✓ Example: **echo -e "One Two\nOne Two Three\nOne Two Three Four" | awk 'NF > 2’**
- **NR**
  - ✓ represents the number of the current record
  - ✓ Example: **echo -e "One Two\nOne Two Three\nOne Two Three Four" | awk 'NR < 3'**

7. Built-in Variables

Standard AWK variables

- **$0**
  - ✓ represents the entire input record
  - ✓ Example: **awk '{print $0}' marks.txt**

- **$n**
  - ✓ represents the nth field in the current record
  - ✓ Example: **awk '{print $3 "\t" $4}' marks.txt**

8. Operators

- **Arithmetic Operators**
  - **Addition**

    ```
    awk 'BEGIN { a = 50; b = 20; print "(a + b) = ", (a + b) }’
    ```

  - **Subtraction**

    ```
    awk 'BEGIN { a = 50; b = 20; print "(a - b) = ", (a - b) }’
    ```

  - **Multiplication**

    ```
    awk 'BEGIN { a = 50; b = 20; print "(a * b) = ", (a * b) }’
    ```

  - **Division**

    ```
    awk 'BEGIN { a = 50; b = 20; print "(a / b) = ", (a / b) }’
    ```

  - **Modulus**

    ```
    awk 'BEGIN { a = 50; b = 20; print "(a % b) = ", (a % b) }'
    ```

8. Operators

- **Increment and Decrement Operators**
  - **Pre-Increment/Decrement**
    - ✓ awk 'BEGIN { a = 10; b = ++a; printf "a = %d, b = %d\n", a, b }’
    - ✓ awk 'BEGIN { a = 10; b = --a; printf "a = %d, b = %d\n", a, b }’
  - **Post-Increment/Decrement**
    - ✓ awk 'BEGIN { a = 10; b = a++; printf "a = %d, b = %d\n", a, b }’
    - ✓ awk 'BEGIN { a = 10; b = a--; printf "a = %d, b = %d\n", a, b }'

8. Operators

- **Assignment Operators**
  - **Simple Assignment (=)**

    awk 'BEGIN { name = "Jerry"; print "My name is", name }’

  - **Shorthand**
    - ✓ awk 'BEGIN { cnt = 10; cnt += 10; print "Counter =", cnt }’
    - ✓ awk 'BEGIN { cnt = 100; cnt -= 10; print "Counter =", cnt }’
    - ✓ awk 'BEGIN { cnt = 10; cnt *= 10; print "Counter =", cnt }’
    - ✓ awk 'BEGIN { cnt = 100; cnt /= 5; print "Counter =", cnt }’
    - ✓ awk 'BEGIN { cnt = 100; cnt %= 8; print "Counter =", cnt }'

8. Operators

- **Relational Operators**
  - **Equal to**

    ```
    awk 'BEGIN { a = 10; b = 10; if (a == b) print "a == b" }’
    ```

  - **Not Equal to**

    ```
    awk 'BEGIN { a = 10; b = 20; if (a != b) print "a != b" }’
    ```

  - **Less Than**

    ```
    awk 'BEGIN { a = 10; b = 20; if (a < b) print "a  < b" }’
    ```

8. Operators

- **Relational Operators**
  - **Less Than or Equal to**

    awk 'BEGIN { a = 10; b = 10; if (a <= b) print "a <= b" }’
  - **Greater Than**

    awk 'BEGIN { a = 10; b = 20; if (b > a) print "b > a" }’
  - **Greater Than or Equal to**

    awk 'BEGIN { a = 10; b = 20; if (b >= a) print "b > a" }’

8. Operators

- **Logical Operators**
  - **Logical AND (&&)**

    expr1 && expr2

    awk 'BEGIN {num = 5; if (num >= 0 && num <= 7) printf "%d is in octal format\n", num }’
  - **Logical OR (||)**

    expr1 || expr2

    awk 'BEGIN {ch = "\n"; if (ch == " " || ch == "\t" || ch == "\n") print "Current character is whitespace." }’
  - **Logical NOT (!)**

    ! expr1

    awk 'BEGIN { name = ""; if (! length(name)) print "name is empty string." }'

8. Operators

- **String Concatenation Operator**
  - Space is a string concatenation operator that merges two strings
  - Example: **awk 'BEGIN { str1 = "Hello, "; str2 = "World"; str3 = str1 str2; print str3 }'**

- **Array Membership Operator**
  - It is represented by **in**
  - Example: **awk 'BEGIN {arr[0] = 1; arr[1] = 2; arr[2] = 3; for (i in arr) printf "arr[%d] = %d\n", i, arr[i]}'**

9. Control Flow

- **If statement**
  - Syntax:

```
if (condition) {
    action-1
    action-2
    .
    .
    action-n
}
```

  - Example: **awk 'BEGIN {num = 10; if (num % 2 == 0) printf "%d is even number.\n", num }'**

9. Control Flow

- **If Else Statement**
  - Syntax:

```
if (condition)
    action-1
else
    action-2
```

  - Example: **awk 'BEGIN {num = 11; if (num % 2 == 0) printf "%d is even number.\n", num; else printf "%d is odd number.\n", num}'**

9. Control Flow

- **If-Else-If Ladder**
  - Can easily create an if-else-if ladder by using multiple if-else statements
  - Example:

```
awk 'BEGIN {
   a = 30;
   if (a==10)
   print "a = 10";
   else if (a == 20)
   print "a = 20";
   else if (a == 30)
   print "a = 30";
}'
```

10. Loops

- Loops are used to execute a set of actions in a repeated manner
- The loop execution continues as long as the loop condition is true
- **For Loop**
  - Syntax:

    for (initialization; condition; increment/decrement)

    action

  - Example:

    awk 'BEGIN { for (i = 1; i <= 5; ++i) print i }'

10. Loops

- **While Loop**
  - Syntax:

    while (condition)

    action

  - Example:

    awk 'BEGIN {i = 1; while (i < 6) { print i; ++i } }'

11. Pretty Printing

- **printf function**
  - Syntax: **printf fmt, expr-list**
  - **fmt** is a string of format specifications and constants
  - **expr-list** is a list of arguments corresponding to format specifiers

- **Escape Sequences**
  - **New Line**

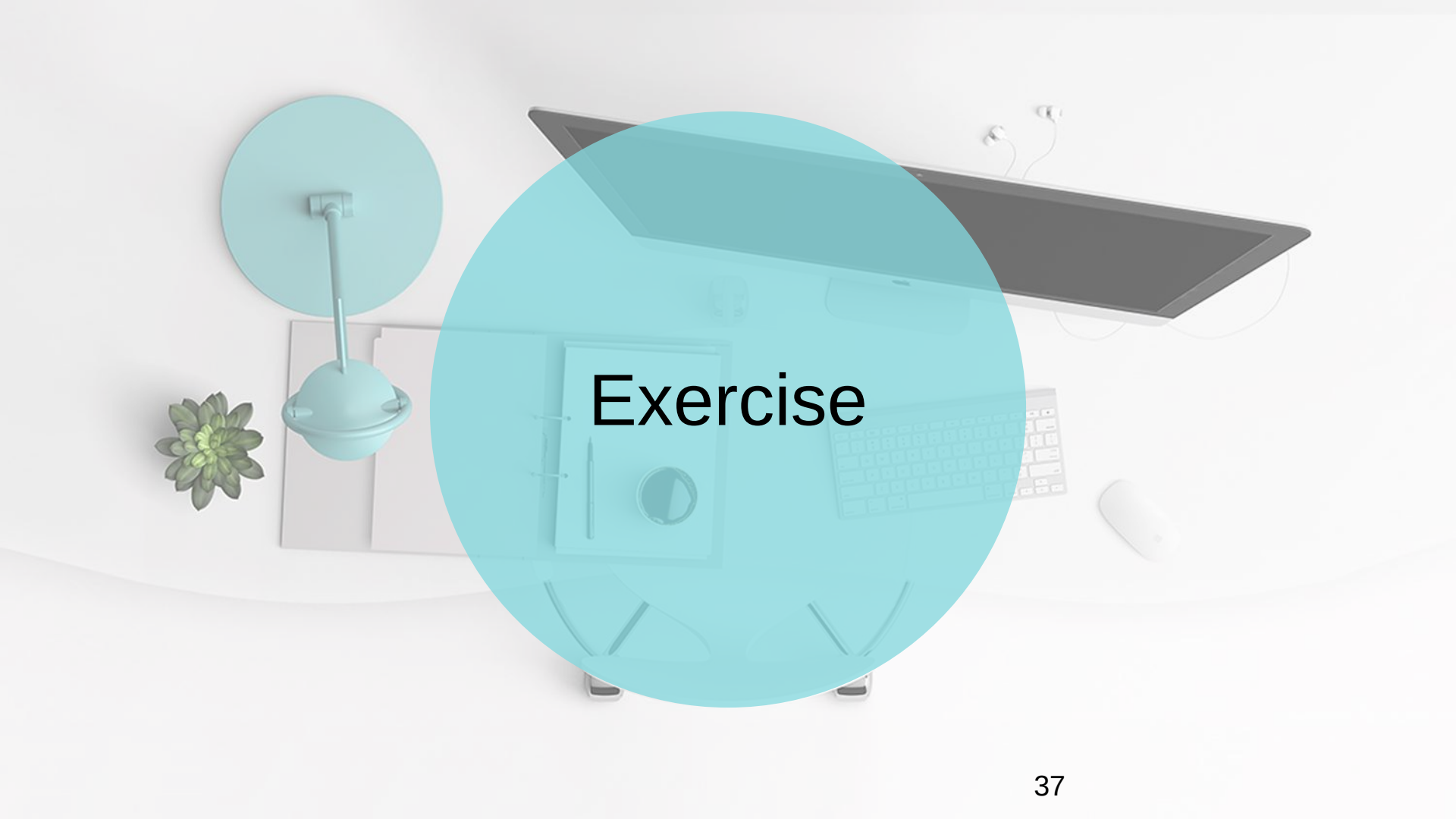
    awk 'BEGIN { printf "Hello\nWorld\n" }'
  - **Horizontal Tab**

    awk 'BEGIN { printf "Sr No\tName\tSub\tMarks\n" }'

11. Pretty Printing

- **Escape Sequences**
  - **Format Specifier**
    - ✓ **%c:** prints a single character
    - ✓ **%d** and **%i:** prints only the integer part of a decimal number
    - ✓ **%f:** prints a floating point number of the form [-]ddd.dddddd
    - ✓ **%g and %G:** uses %e or %f conversion, whichever is shorter, with non-significant zeros suppressed
    - ✓ **%s:** prints a character string

      awk 'BEGIN { printf "Name = %s\n", "Sherlock Holmes" }'



Exercise

Q&A