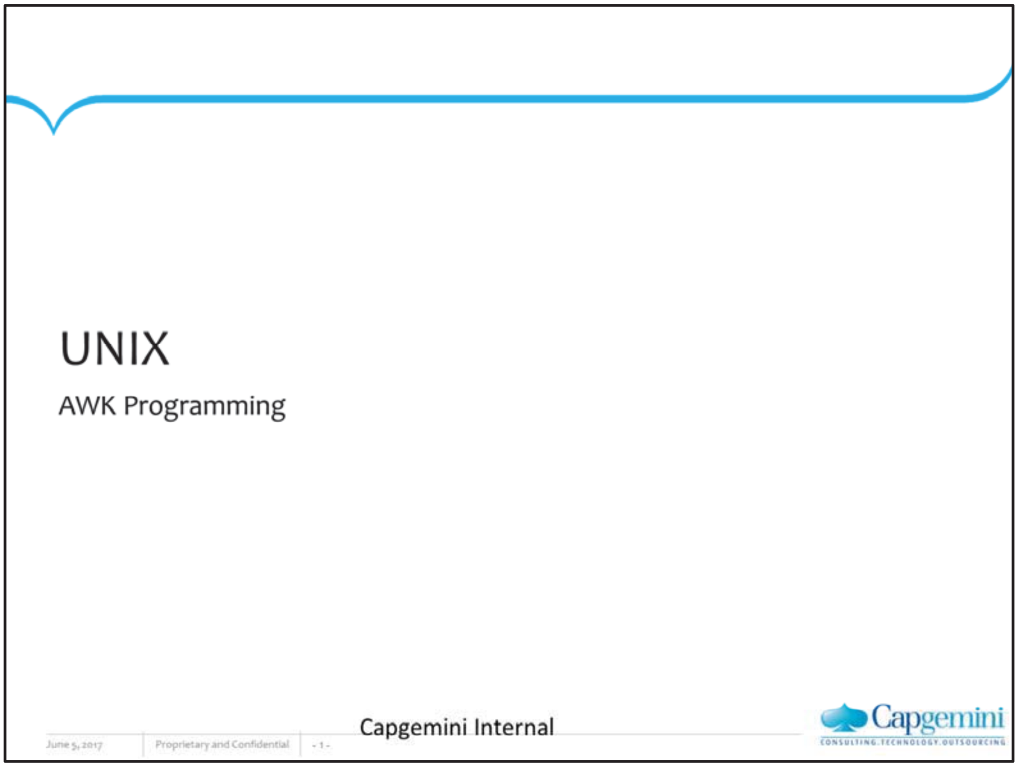
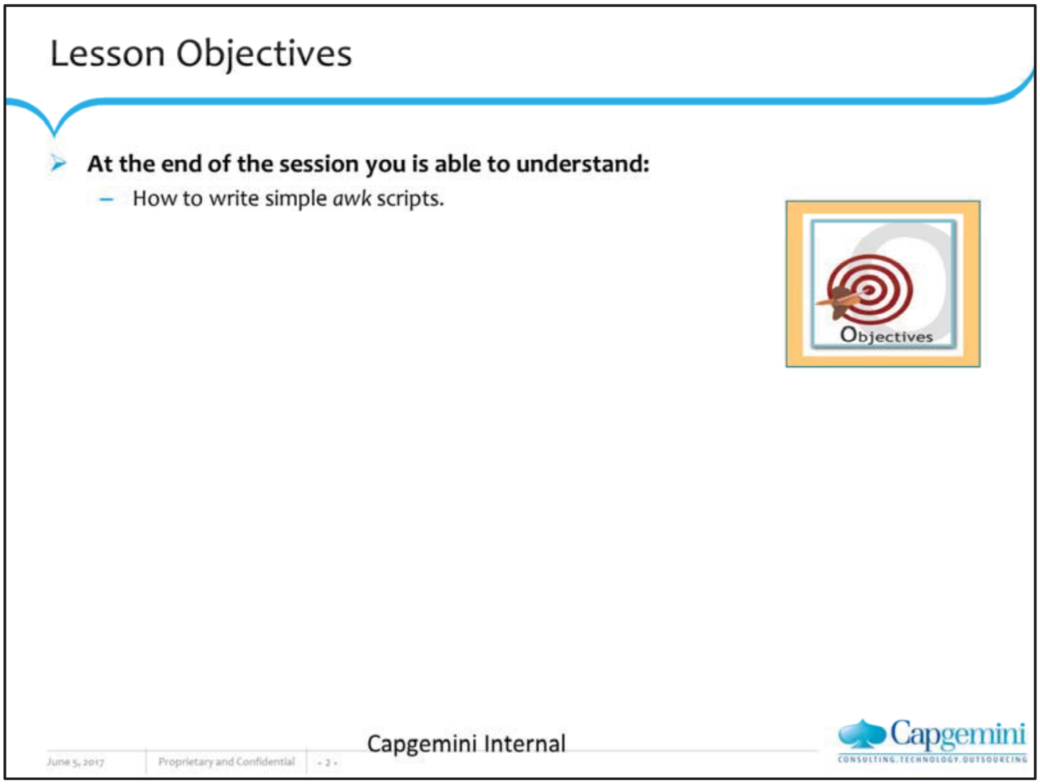


UNIX

AWK Programming

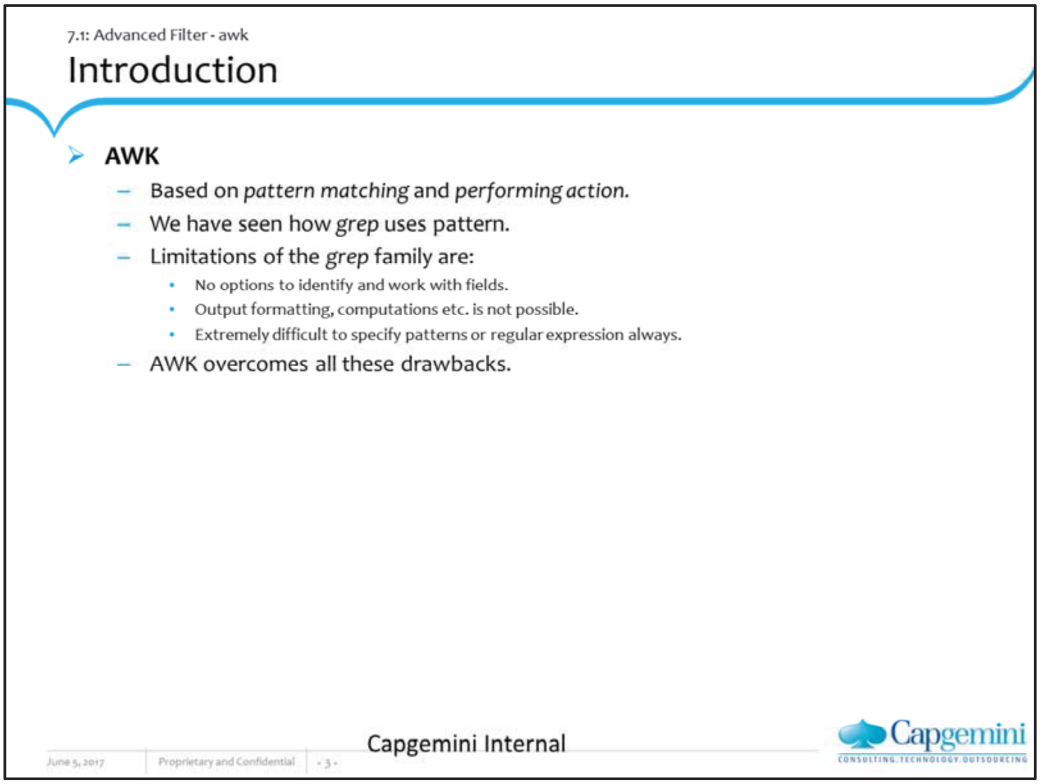
June 5, 2017 | Proprietary and Confidential | - 1 -

Capgemini Internal



Lesson Objectives

- **At the end of the session you is able to understand:**
  - How to write simple *awk* scripts.



7.1: Advanced Filter - awk

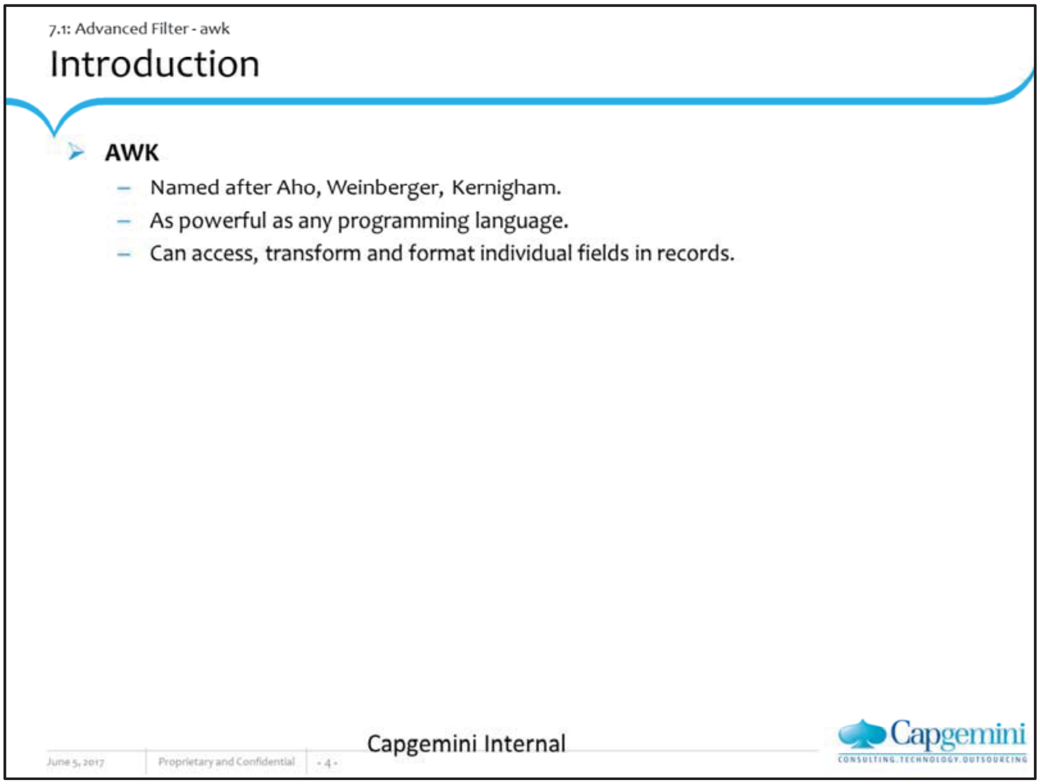
Introduction

- **AWK**
  - Based on *pattern matching* and *performing action.*
  - We have seen how *grep* uses pattern.
  - Limitations of the *grep* family are:
    - No options to identify and work with fields.
    - Output formatting, computations etc. is not possible.
    - Extremely difficult to specify patterns or regular expression always.
  - AWK overcomes all these drawbacks.

June 5, 2017 | Proprietary and Confidential | - 3 -

Capgemini Internal

The *awk* command, named after its authors Aho, Weinberger and Kernighan, is one of the most powerful utilities for text manipulation. It combines features of many other filters. It can access, transform and format individual fields in a record – it is also called as a report writer. It can accept regular expressions for pattern matching, has “C” type programming constructs, variables and in-built functions. In fact, awk is nearly as powerful as any other programming language. However, awk programs are generally slow, if any alternative commands are available to do the same use them rather than using AWK.



7.1: Advanced Filter - awk

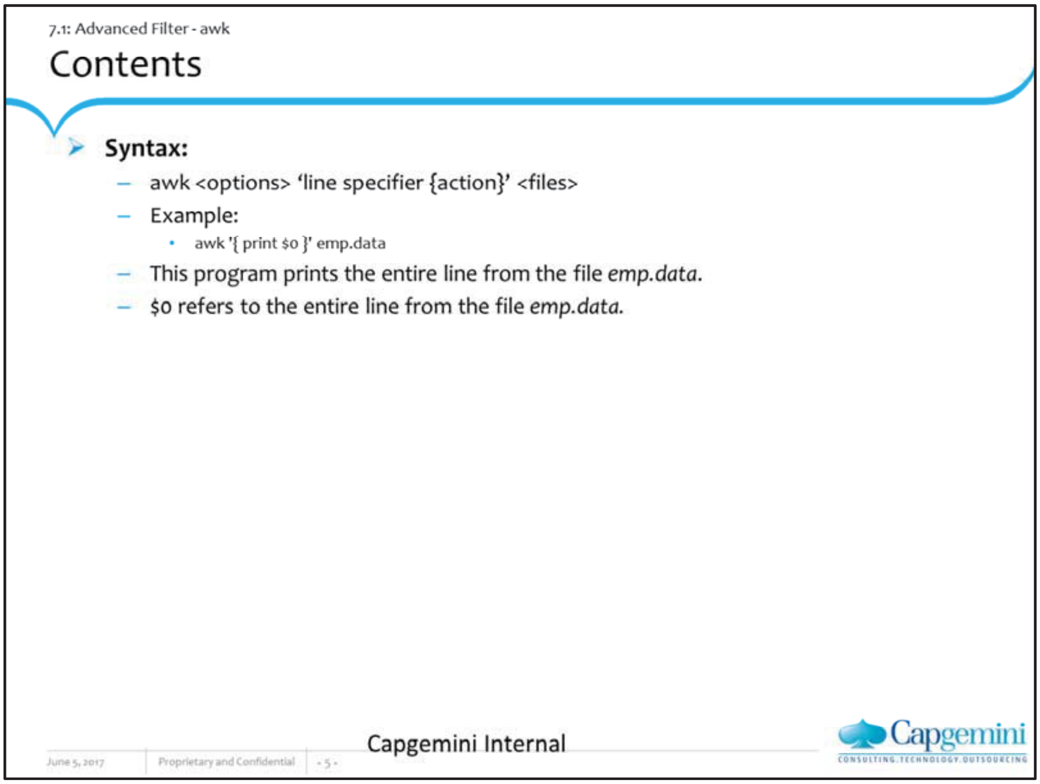
Introduction

- **AWK**
  - Named after Aho, Weinberger, Kernigham.
  - As powerful as any programming language.
  - Can access, transform and format individual fields in records.

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 4 -

Filters accept some data as input, perform some manipulation on it and produce some output. Some simple filters have been discussed in the previous chapters. This chapter discusses advanced filter –*awk*.



7.1: Advanced Filter - awk

Contents

- **Syntax:**
  - awk <options> ‘line specifier {action}’ <files>
  - Example:
    - awk '{ print $0 }' emp.data
  - This program prints the entire line from the file *emp.data.*
  - $0 refers to the entire line from the file *emp.data.*

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 5 -

Advanced Filter awk.

The general syntax of the awk command is:

```
awk <options> ‘line specifier {action}’ <file(s)>
```

Simple awk Filtering

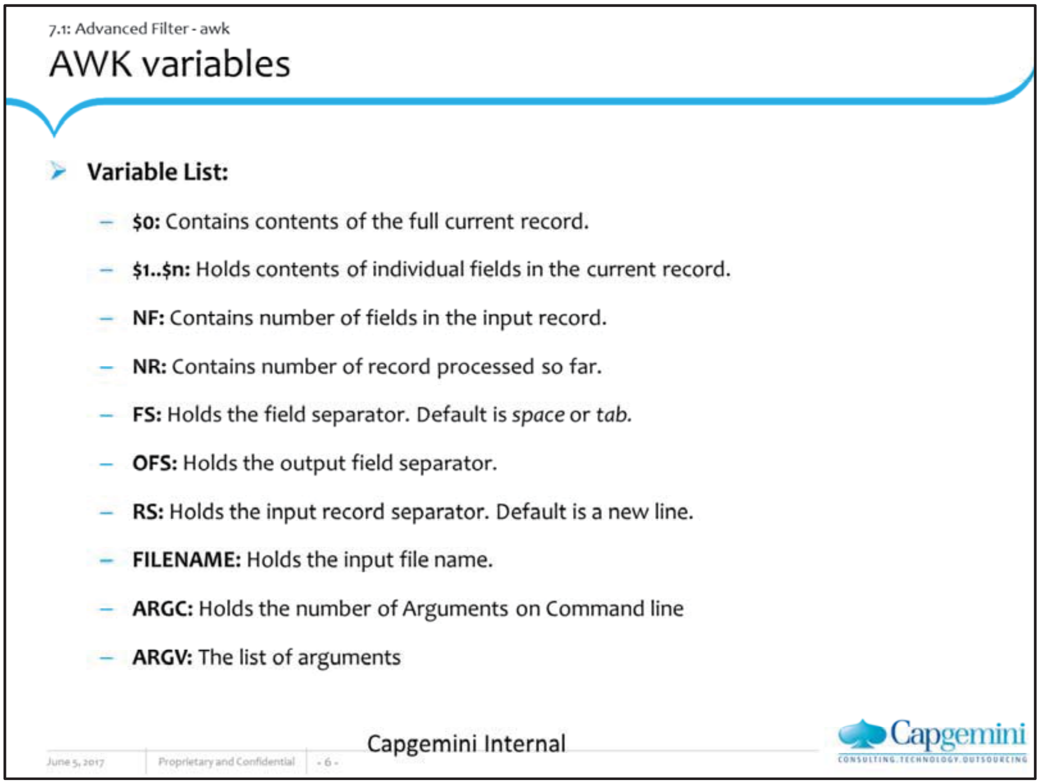
Following is an example of a simple awk command:

It prints all lines from file *books.lst* in which pattern ‘Computer’ is found.

```
$ awk '/Computer/ {print}' books.lst
```

Output:

```
1001|Learning Unix            |Computers    |01/01/1998| 575
1003|XML Unleashed            |Computers    |20/02/2000| 398
1004|Unix Device Drivers      |Computers    |09/08/1995| 650
1007|Unix Shell Programming   |Computers    |03/02/1993| 536
```



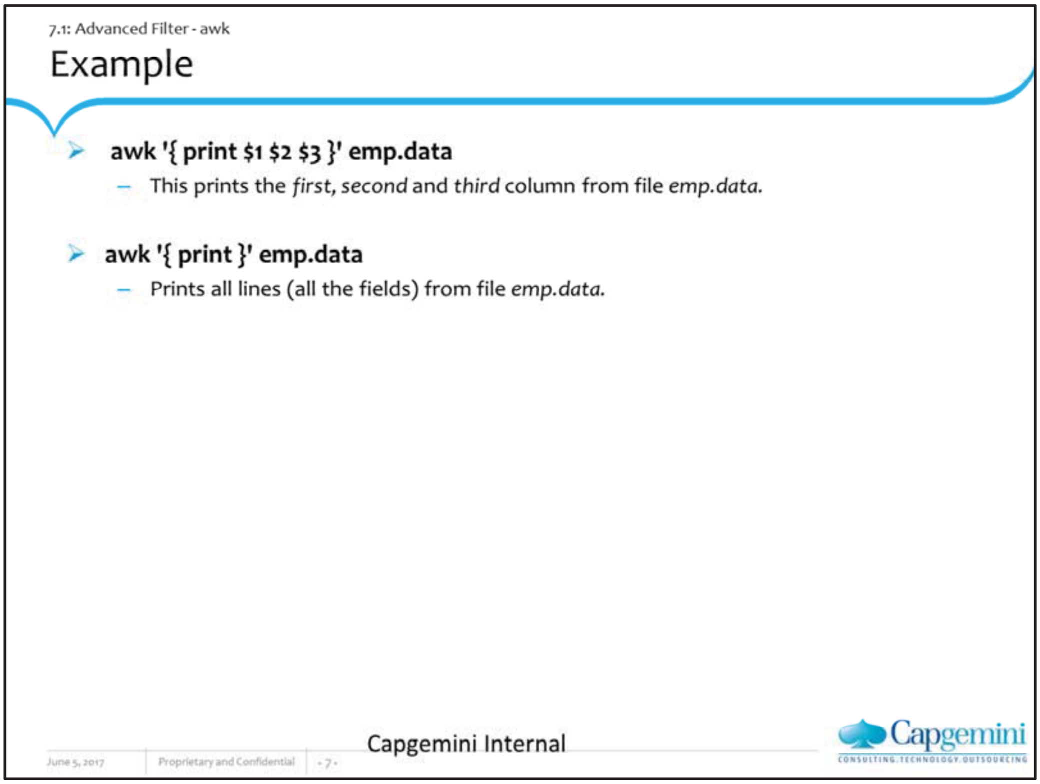
7.1: Advanced Filter - awk

AWK variables

- **Variable List:**
  - **$0:** Contains contents of the full current record.
  - **$1..$n:** Holds contents of individual fields in the current record.
  - **NF:** Contains number of fields in the input record.
  - **NR:** Contains number of record processed so far.
  - **FS:** Holds the field separator. Default is *space* or *tab.*
  - **OFS:** Holds the output field separator.
  - **RS:** Holds the input record separator. Default is a new line.
  - **FILENAME:** Holds the input file name.
  - **ARGC:** Holds the number of Arguments on Command line
  - **ARGV:** The list of arguments

June 5, 2017 | Proprietary and Confidential | - 6 -

Capgemini Internal



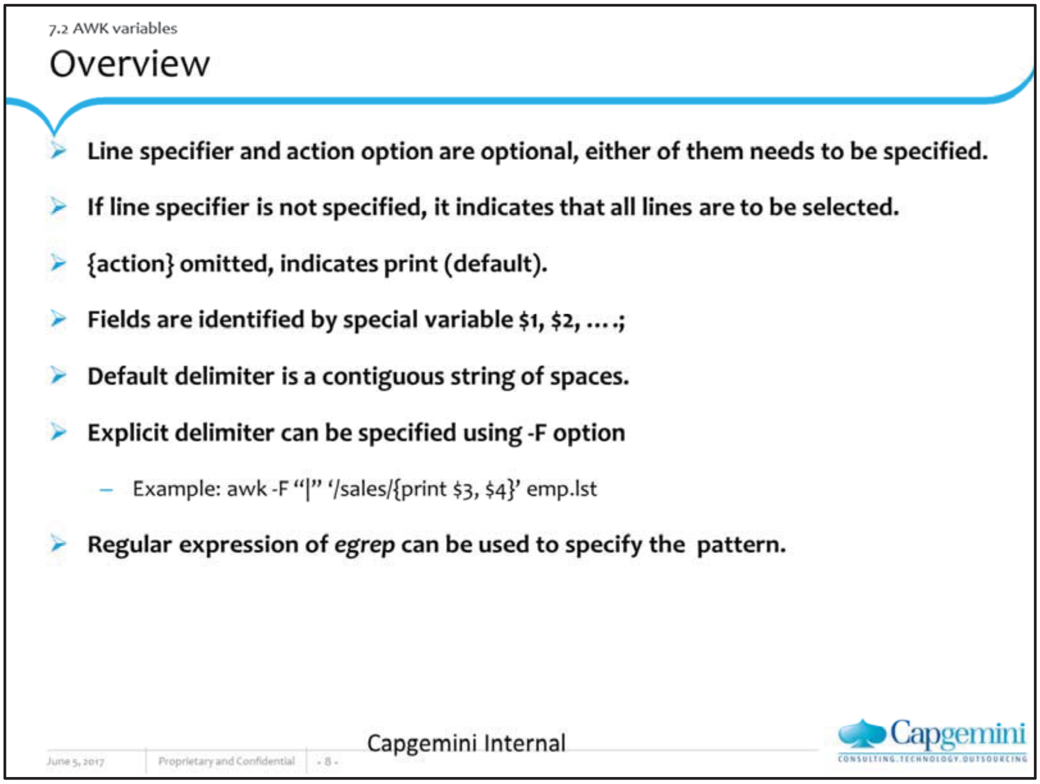
7.1: Advanced Filter - awk

Example

- **awk '{ print $1 $2 $3 }' emp.data**
  - This prints the *first*, *second* and *third* column from file *emp.data.*

- **awk '{ print }' emp.data**
  - Prints all lines (all the fields) from file *emp.data.*

Capgemini Internal



7.2 AWK variables

Overview

- Line specifier and action option are optional, either of them needs to be specified.
- If line specifier is not specified, it indicates that all lines are to be selected.
- {action} omitted, indicates print (default).
- Fields are identified by special variable $1, $2, ….;
- Default delimiter is a contiguous string of spaces.
- Explicit delimiter can be specified using -F option
  - Example: awk -F "|" '/sales/{print $3, $4}' emp.lst
- Regular expression of *egrep* can be used to specify the pattern.

June 5, 2017 | Proprietary and Confidential | - 8 - Capgemini Internal

The line specifier uses a context address to specify the lines that need to be taken up for processing in the action section. If the line specifier is missing, then the action is applicable to all lines of the file.

In the action part, statement {print} indicates that selected lines are to be printed. The statement {print} is equivalent to {print $0} - $0 being the variable for the entire line. The awk command is also capable of breaking each line into fields – each field is identified as $1, $2 etc.

For the purpose of identification of fields, awk uses a contiguous string of spaces as field delimiter. However, it is possible to use a different delimiter. This is specified using the –F option in awk.

$ awk -F"|" '/Computer/ {print $2,$5}' books.lst
Output:

```
Learning Unix           575
XML Unleashed           398
Unix Device Drivers      650
Unix Shell Programming    536
```

For pattern matching, awk uses regular expressions of *egrep* variety.
$ awk -F"|" '/XML|Unix/ {print $2, $5}' books.lst
Output:

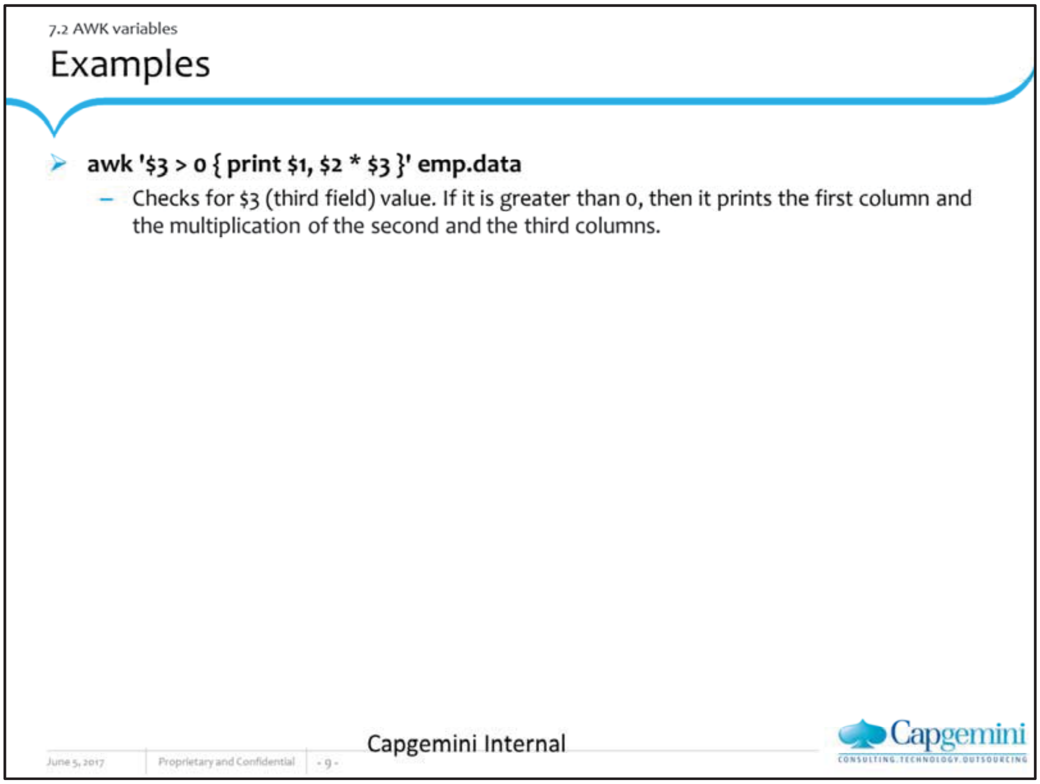
```
Learning Unix            575
XML Unleashed           398
Unix Device Drivers      650
XML Applications        630
Unix Shell Programming    536
```

It is possible to specify line numbers in file using the inbuilt NR variable. Also, awk can use the C-like *printf* statement to format the output.

$ awk -F"|" '$1=="1002" {printf "%2d,%-20s",NR,$2}' books.lst
Output:

```
 2,Moby Dick
```

The –f option of awk is useful if you wish to store the line specifier or action in a separate file.



7.2 AWK variables

Examples

- **awk '$3 > 0 { print $1, $2 * $3 }' emp.data**
  - Checks for $3 (third field) value. If it is greater than 0, then it prints the first column and the multiplication of the second and the third columns.

June 5, 2017 | Proprietary and Confidential | - 9 -

Capgemini Internal

The logical operators || (or) and && (and) are used by the awk command to combine conditions in the line specifier. A relational operator can be used in the line specifier, also in the action component.

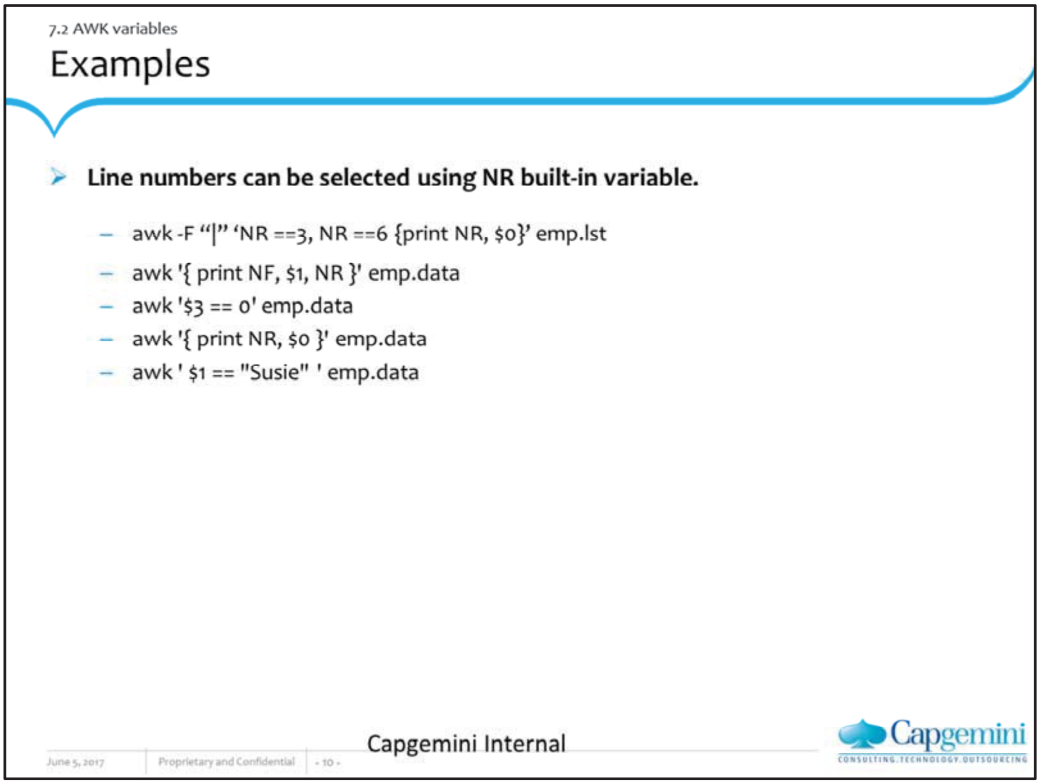
The operators == (equal) and != (not equal) can handle only fixed length strings and not regular expressions. To match regular expressions, ~ (match) and !~ (negate) are used. The characters ^ and $ can be used for looking for a pattern in the beginning and end of field.

To work with numbers, operators like < (less than), > (greater than), <= (less than or equal), >= (greater than or equal), == (equal) and != (not equal) can be used.
It is possible to perform computations on numbers using C like arithmetic operators (+, -, *, /, %, ++, --, += etc).

```
$ awk -F"|" '/Unix/ && $5 < 600 {printf "%s,%d\n",$2,$5}' books.lst
```

Output:

```
Learning Unix          ,575
Unix Shell Programming   ,536
$ awk -F"|" '$2=="Learning Unix"' books.lst
$ awk -F"|" '$2~/Learning Unix/' books.lst
1001|Learning Unix          |Computers     |01/01/1998| 575
$ awk -F"|" '$5<500 {
> cnt=cnt+1
> printf "%d %s\n",cnt,$2}' books.lst
1 Moby Dick
2 XML Unleashed
```



7.2 AWK variables

Examples

- Line numbers can be selected using NR built-in variable.
  - awk -F “|” ‘NR ==3, NR ==6 {print NR, $0}’ emp.lst
  - awk '{ print NF, $1, NR }' emp.data
  - awk '$3 == 0' emp.data
  - awk '{ print NR, $0 }' emp.data
  - awk ' $1 == "Susie" ' emp.data

June 5, 2017 | Proprietary and Confidential | - 10 -

Capgemini Internal

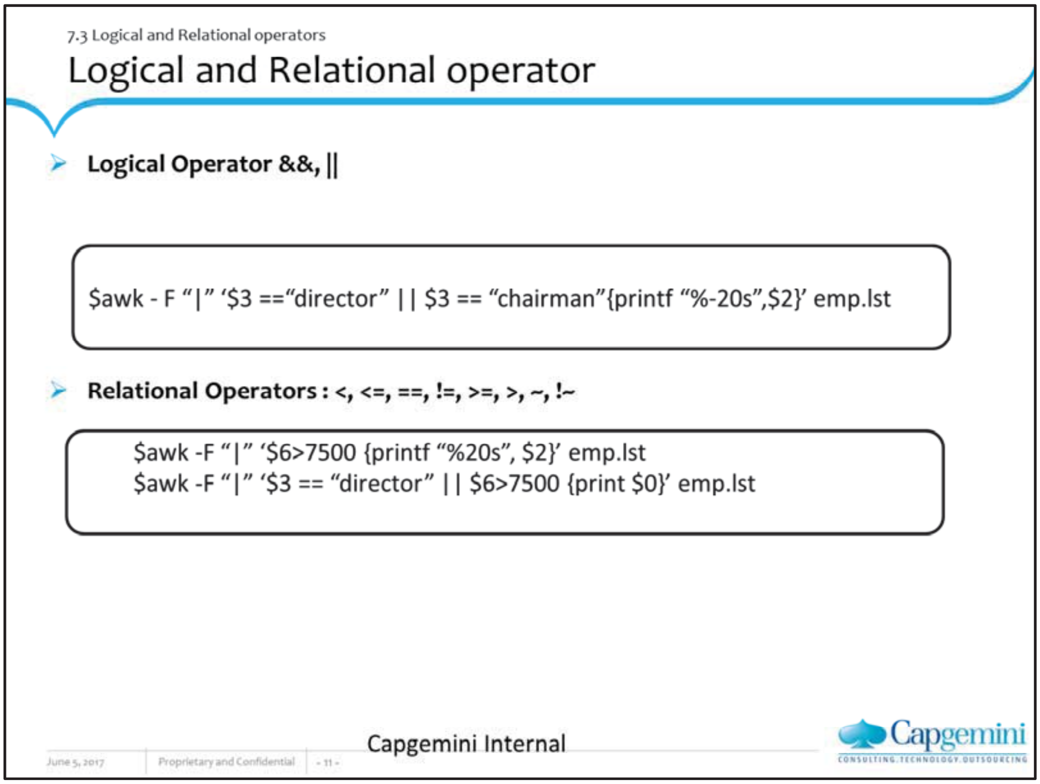
awk -F “|” ‘NR ==3, NR ==6 {print NR, $0}’ emp.lst
In this example, NR represents record number. It prints records from *third record* to *sixth* record.
-F is use to specify field separator.

awk '{ print NF, $1, NR }' emp.data
This prints number of fields, contents of field 1 and record number for all records.

awk '$3 == 0' emp.data
It prints all lines in which the value in the third field is 0.

awk '{ print NR, $0 }' emp.data
It will print record number and record for all records

awk ' $1 == "Susie" ' emp.data
It prints all lines in which the value in the first field is Susie.



7.3 Logical and Relational operators

Logical and Relational operator

- **Logical Operator &&, ||**

```
$awk - F “|” ‘$3 ==“director” || $3 == “chairman”{printf “%-20s”,$2}’ emp.lst
```

- **Relational Operators : <, <=, ==, !=, >=, >, ~, !~**

```
$awk -F “|” ‘$6>7500 {printf “%20s”, $2}’ emp.lst
$awk -F “|” ‘$3 == “director” || $6>7500 {print $0}’ emp.lst
```

June 5, 2017 | Proprietary and Confidential | - 11 -

Capgemini Internal

The logical operators || (or) and && (and) are used by the awk command to combine conditions in the line specifier. A relational operator can be used in the line specifier, also in the action component.

The operators == (equal) and != (not equal) can handle only fixed length strings and not regular expressions. To match regular expressions, ~ (match) and !~ (negate) are used. The characters ^ and $ can be used for looking for a pattern in the beginning and end of field.

For working with numbers, operators like < (less than), > (greater than), <= (less than or equal), >= (greater than or equal), == (equal) and != (not equal) can be used.

It is possible to perform computations on numbers using C like arithmetic operators (+, -, *, /, %, ++, --, += etc).

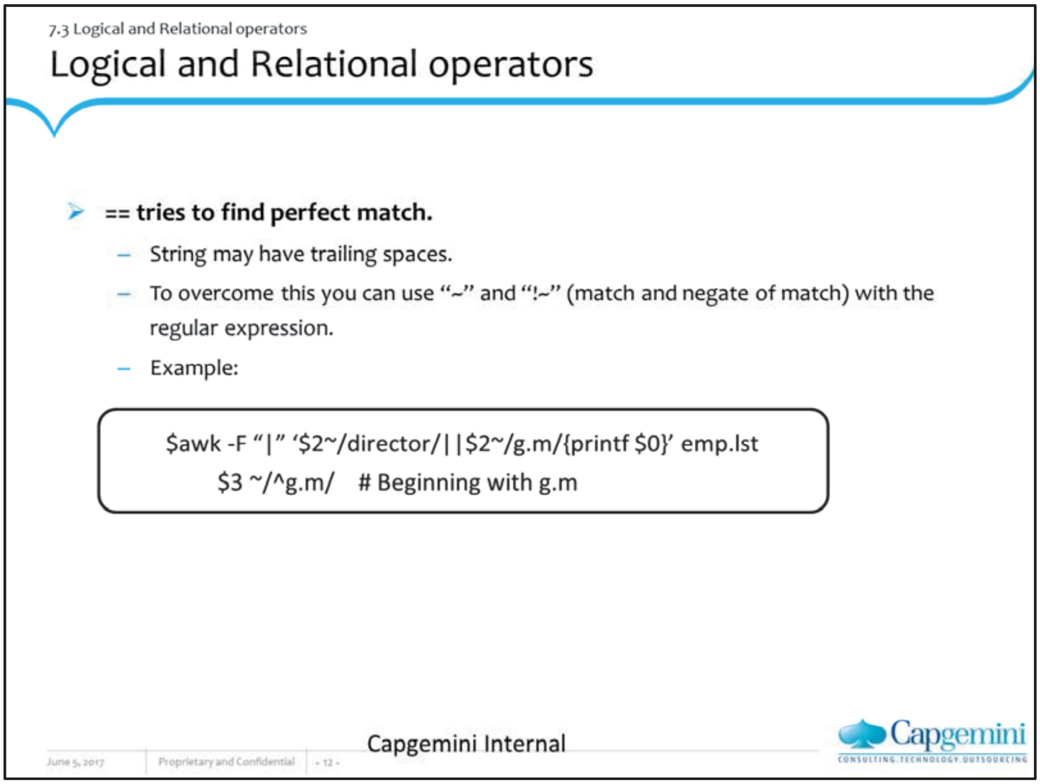
Print all lines with Unix pattern in the line and value of 5 th field should be < 600:

$ awk -F"|" '/Unix/ && $5 < 600 {printf "%s,%d\n",$2,$5}' books.lst

Output

```
Learning Unix                ,575
Unix Shell Programming   ,536
$ awk -F"|" '$2=="Learning Unix"' books.lst
$ awk -F"|" '$2~/Learning Unix/' books.lst
1001|Learning Unix          |Computers     |01/01/1998| 575
$ awk -F"|" '$5<500 {
> cnt=cnt+1
> printf "%d %s\n",cnt,$2}' books.lst
1 Moby Dick
2 XML Unleashed
```

Example of awk command using relational, logical expressions and computations.



7.3 Logical and Relational operators

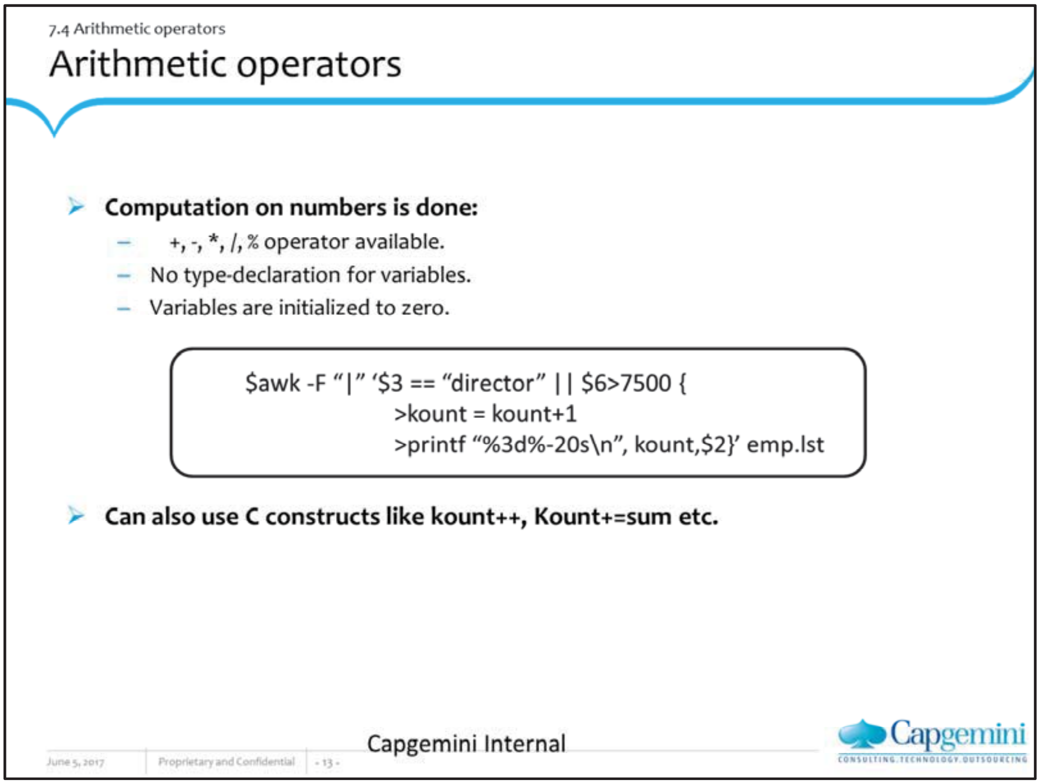
Logical and Relational operators

- **== tries to find perfect match.**
  - String may have trailing spaces.
  - To overcome this you can use “~” and “!~” (match and negate of match) with the regular expression.
  - Example:

```
$awk -F “|” ‘$2~/director/||$2~/g.m/{printf $0}’ emp.lst
      $3 ~/^g.m/    # Beginning with g.m
```

June 5, 2017 | Proprietary and Confidential | - 12 -

Capgemini Internal



7.4 Arithmetic operators

Arithmetic operators

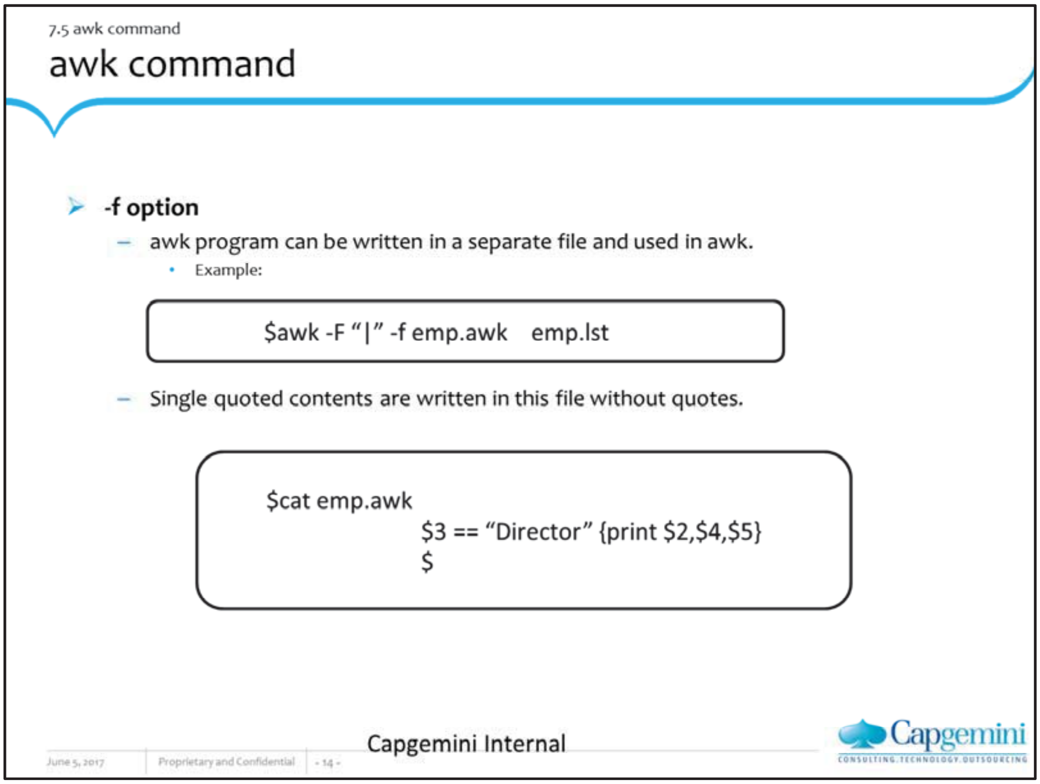
- **Computation on numbers is done:**
  - +, -, *, /, % operator available.
  - No type-declaration for variables.
  - Variables are initialized to zero.

```
$awk -F "|" '$3 == "director" || $6>7500 {
                    >kount = kount+1
                    >printf "%3d%-20s\n", kount,$2}' emp.lst
```

- **Can also use C constructs like kount++, Kount+=sum etc.**

June 5, 2017 | Proprietary and Confidential | - 13 -

Capgemini Internal



7.5 awk command

awk command

- **-f option**
  - awk program can be written in a separate file and used in awk.
    - Example:

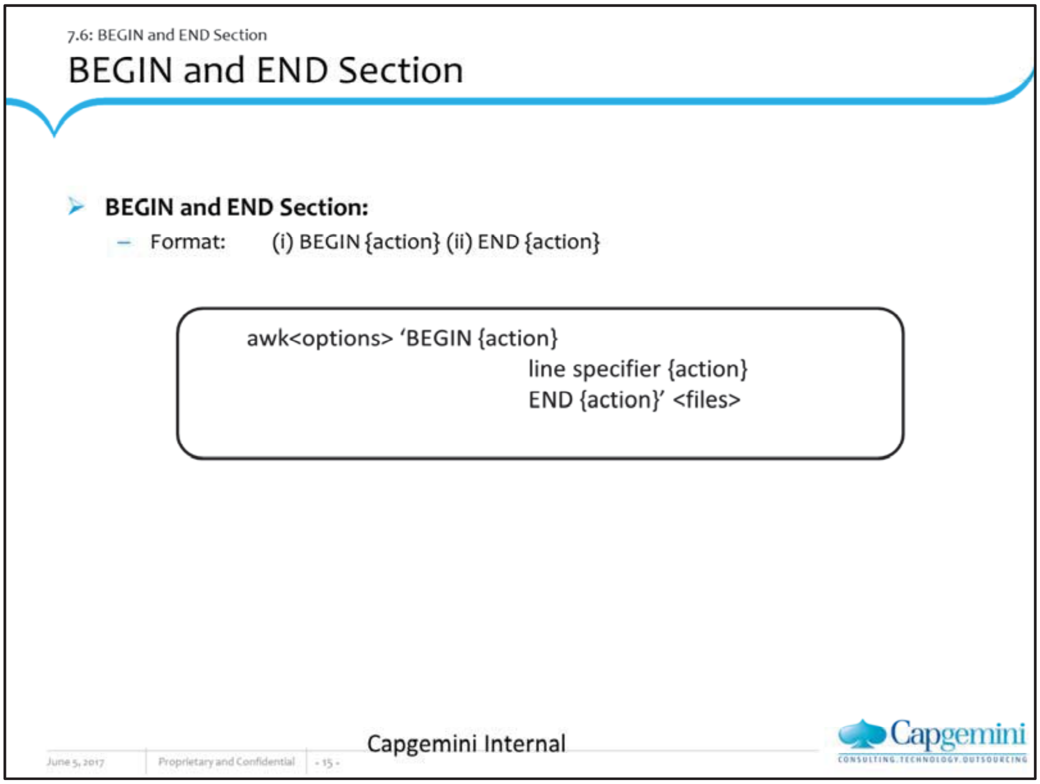
```
$awk -F "|" -f emp.awk    emp.lst
```

  - Single quoted contents are written in this file without quotes.

```
$cat emp.awk
            $3 == "Director" {print $2,$4,$5}
            $
```

June 5, 2017 | Proprietary and Confidential | - 14 -

Capgemini Internal



7.6: BEGIN and END Section

BEGIN and END Section

- **BEGIN and END Section:**
  - Format: (i) BEGIN {action} (ii) END {action}

```
awk<options> 'BEGIN {action}
                                line specifier {action}
                                END {action}' <files>
```

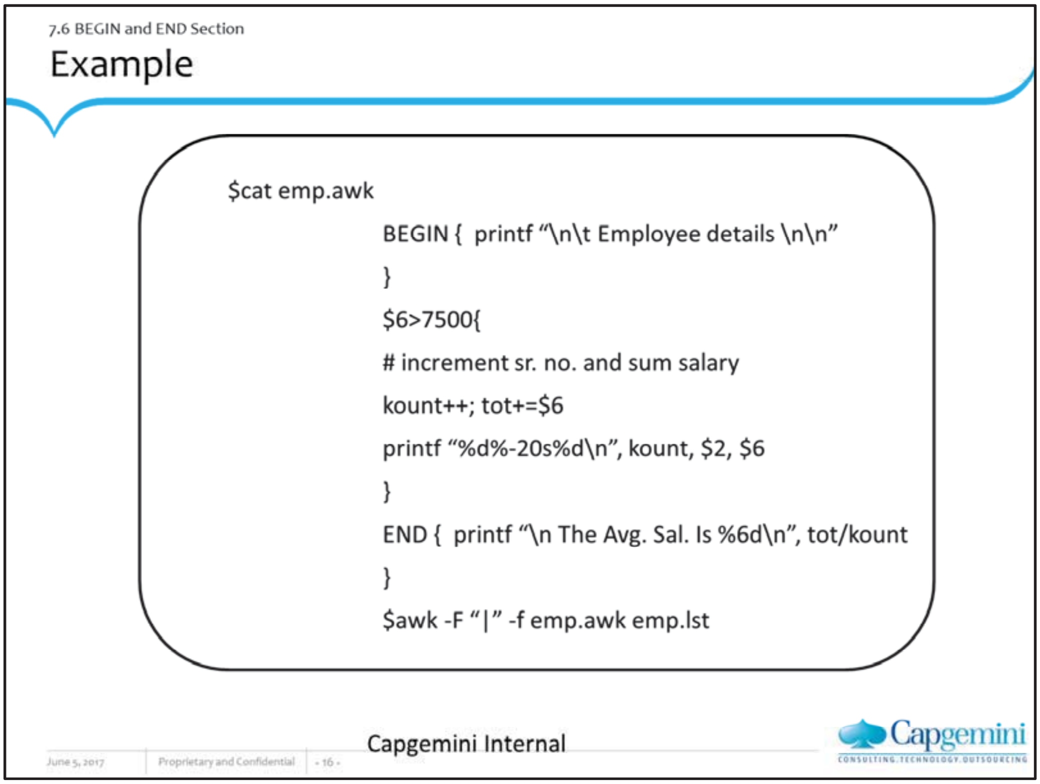
June 5, 2017 | Proprietary and Confidential | - 15 -

Capgemini Internal

<u>BEGIN and END Sections</u>

In case, something is to be printed before processing the first line begins, BEGIN section can be used. Similarly, to print at the end of processing, END section can be used.
These sections are optional. When present, they are delimited by the body of the awk program.

Normally if you want to print any header lines or want to set field separator then use BEGIN section
And If you want to display total or any summarized information at the end of the report then use END section



7.6 BEGIN and END Section

Example

```
$cat emp.awk
            BEGIN {  printf "\n\t Employee details \n\n"
            }
            $6>7500{
            # increment sr. no. and sum salary
            kount++; tot+=$6
            printf "%d%-20s%d\n", kount, $2, $6
            }
            END {  printf "\n The Avg. Sal. Is %6d\n", tot/kount
            }
            $awk -F "|" -f emp.awk emp.lst
```

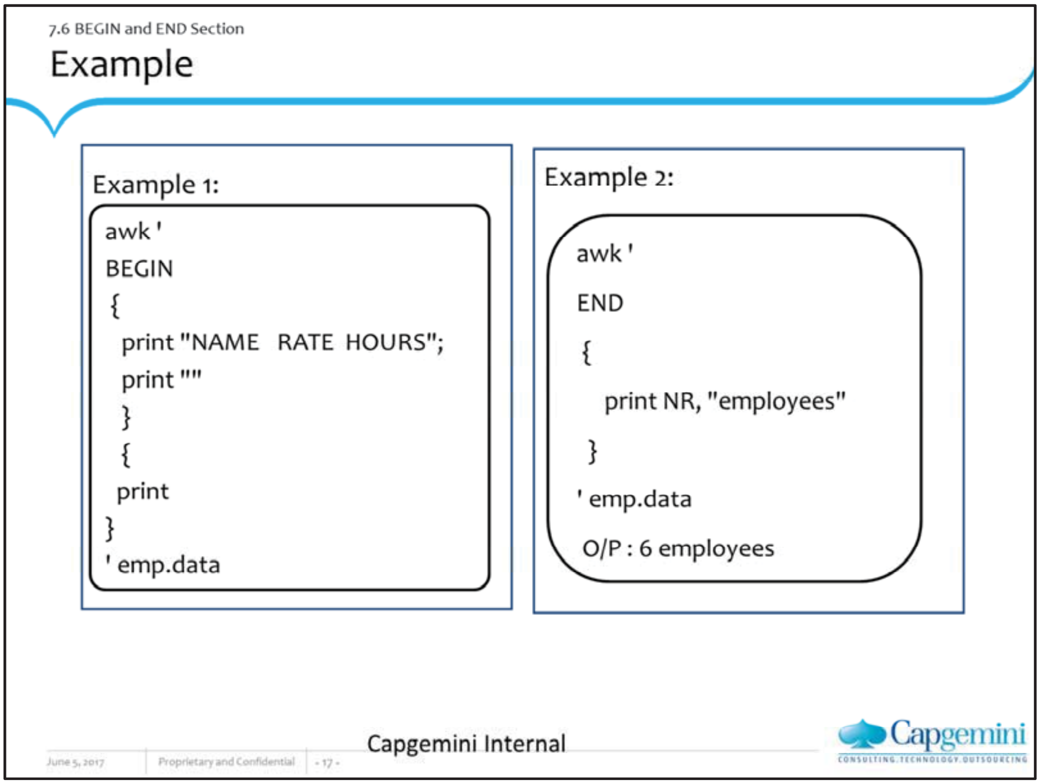
Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 16 -

```
$ cat awk1.awk
BEGIN {
printf "\n\t Unix Book Details \n\n"
}
$2~/Unix/ {
cnt++; tot+=$5
printf "%d %-15s\n",cnt,$2
}
END {
printf "\n Total Cost is %d\n",tot
}
$ awk -F"|" -f awk1.awk books.lst
```

Output:

```
        Unix Book Details
1 Learning Unix
2 Unix Device Drivers
3 Unix Shell Programming
  Total Cost is 1761
```



7.6 BEGIN and END Section

Example

Example 1:

```
awk '
BEGIN
{
  print "NAME  RATE  HOURS";
  print ""
  }
  {
 print
}
' emp.data
```

Example 2:

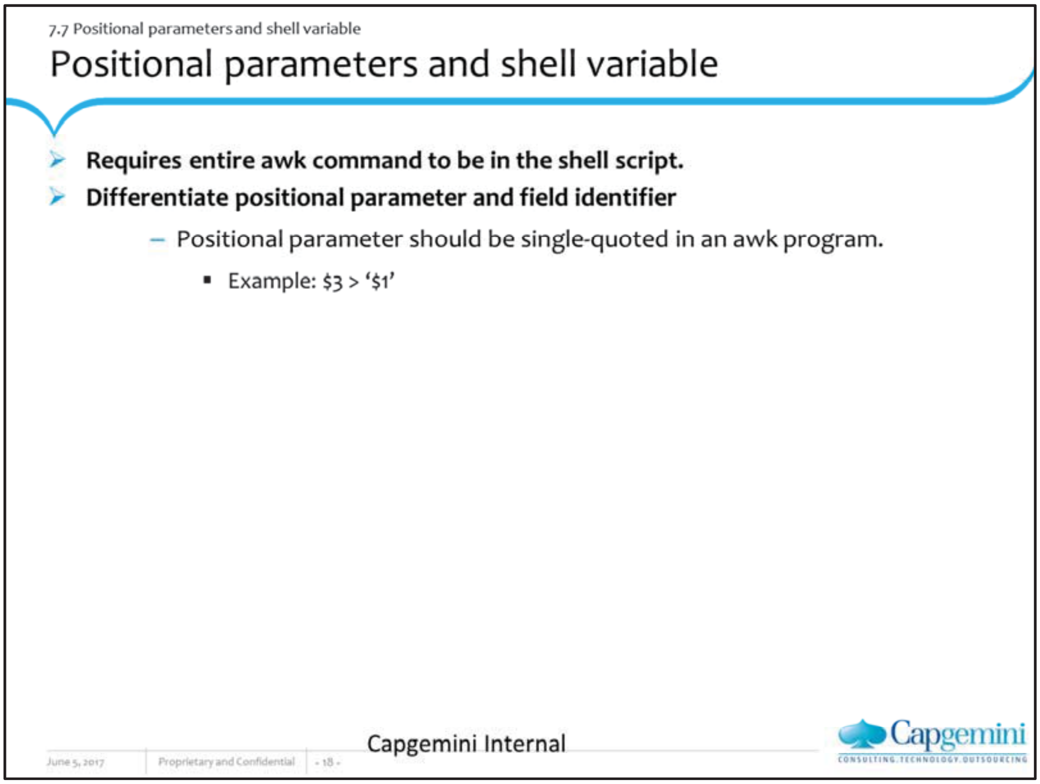
```
awk '
END
{
   print NR, "employees"
 }
' emp.data
 O/P : 6 employees
```

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 17 -

Example 1:
In this example It will perform action enclosed in BEGIN section (It prints headings) and then it will perform print action on all lines in the file emp.dat (because no line specifier is mentioned)

Example 2 :
In this example BEGIN section is not given. The main action part is also empty hence no action is performed on all the records and then it will execute End section which prints last record number which will give you total number of records.



7.7 Positional parameters and shell variable

Positional parameters and shell variable

- **Requires entire awk command to be in the shell script.**
- **Differentiate positional parameter and field identifier**
  - Positional parameter should be single-quoted in an awk program.
    - Example: $3 > '$1'

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 18 -

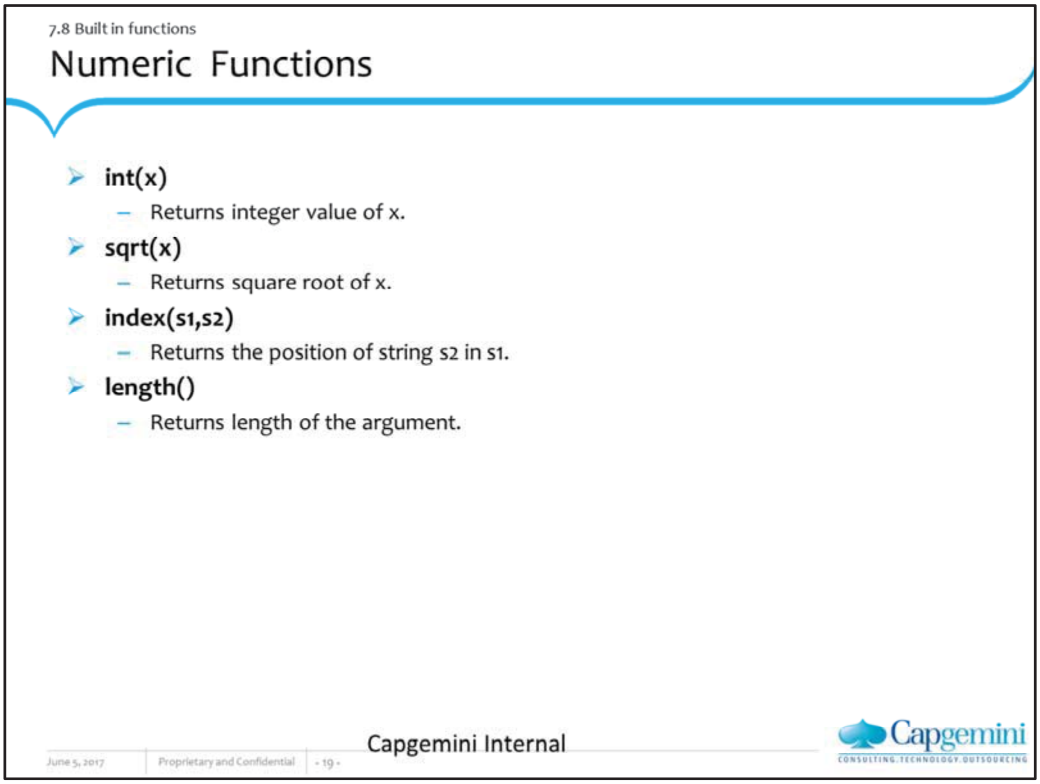
Positional Parameters and Shell Variables

It is possible to store an entire awk command into a file as a shell script, and pass parameters as arguments to the shell script. The shell will identify these arguments as $1, $2 etc based on the order of their occurrence on the command line.

Within awk, since $1, $2 etc. indicate fields of data file, it is necessary to distinguish between the positional parameters and field identifiers. This is done by using single quotes for the positional parameters used by awk.

```
$ cat awk2.awk
awk -F"|" '$5>'$1' {
cnt++
printf "%d %s %d\n",cnt,$2,$5 }
END {
printf "\n No. of books costing more than specified amount is:%d\n",cnt
}' books.lst
$ awk2.awk 600
Output:
1 Unix Device Drivers      650
2 Complete Works:Sherlock H 1290
3 XML Applications        630
No. of books costing more than specified amount is:3
$ awk2.awk 1000
1 Complete Works:Sherlock H 1290
No. of books costing more than specified amount is:1
```

(Example of awk command using positional parameter.)



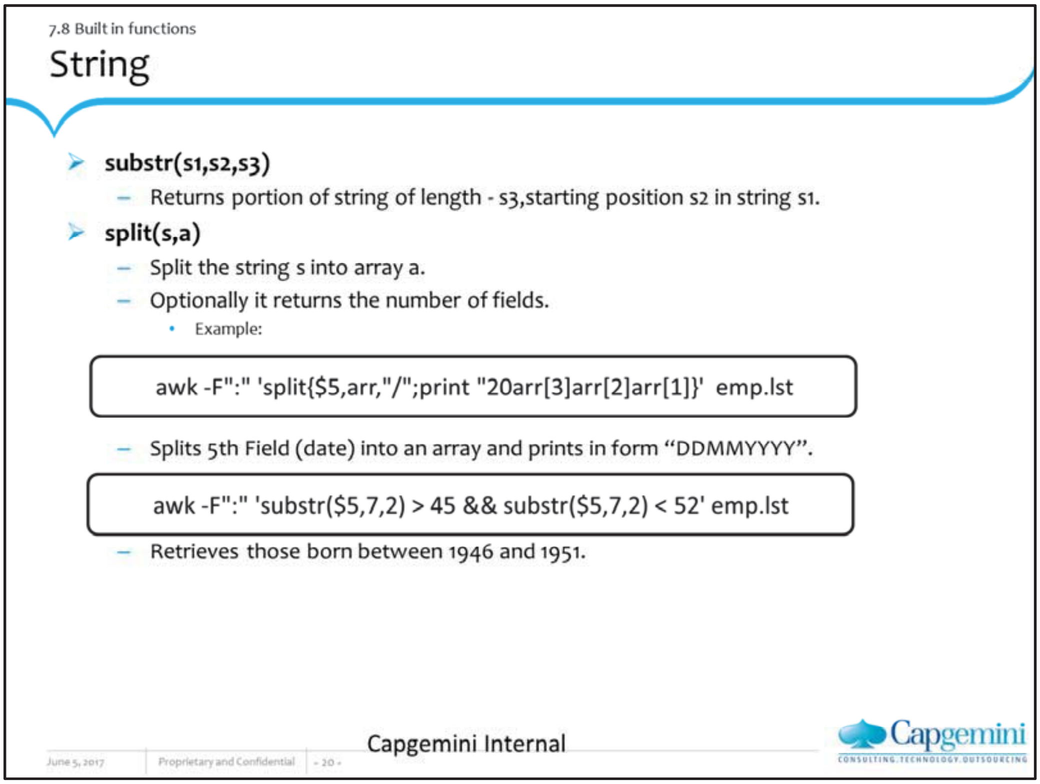
7.8 Built in functions

Numeric Functions

- **int(x)**
  - Returns integer value of x.
- **sqrt(x)**
  - Returns square root of x.
- **index(s1,s2)**
  - Returns the position of string s2 in s1.
- **length()**
  - Returns length of the argument.

June 5, 2017 | Proprietary and Confidential | - 19 -

Capgemini Internal



7.8 Built in functions

String

- **substr(s1,s2,s3)**
  - Returns portion of string of length - s3,starting position s2 in string s1.
- **split(s,a)**
  - Split the string s into array a.
  - Optionally it returns the number of fields.
    - Example:

```
awk -F":" 'split{$5,arr,"/";print "20arr[3]arr[2]arr[1]}'  emp.lst
```

- Splits 5th Field (date) into an array and prints in form "DDMMYYYY".

```
awk -F":" 'substr($5,7,2) > 45 && substr($5,7,2) < 52' emp.lst
```

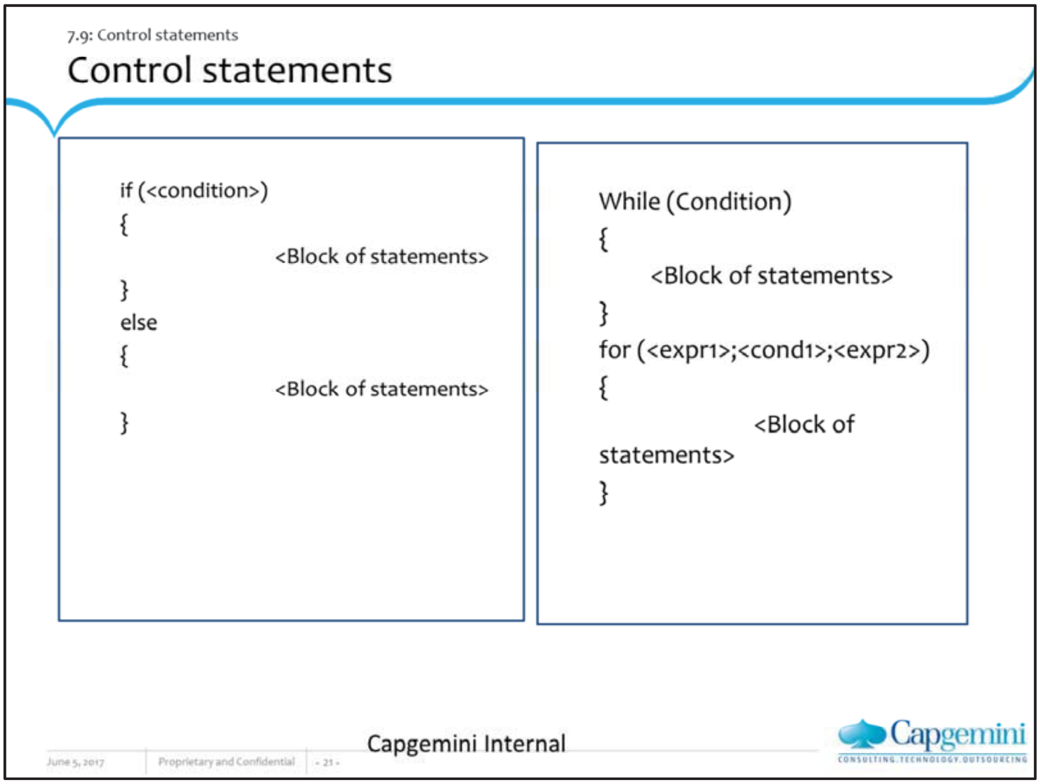
- Retrieves those born between 1946 and 1951.

June 5, 2017 | Proprietary and Confidential | - 20 -

Capgemini Internal

awk -F":" 'length > 1024' emp.lst
Retrieves all lines > 1024.
awk -F":" 'length($2) > 11' emp.lst
Retrieves all lines whose characters in 2nd col < 11.

awk -F":" 'substr($5,7,2) > 45 && substr($5,7,2) < 52' emp.lst
Retrieves those born between 1946 and 1951.



7.9: Control statements

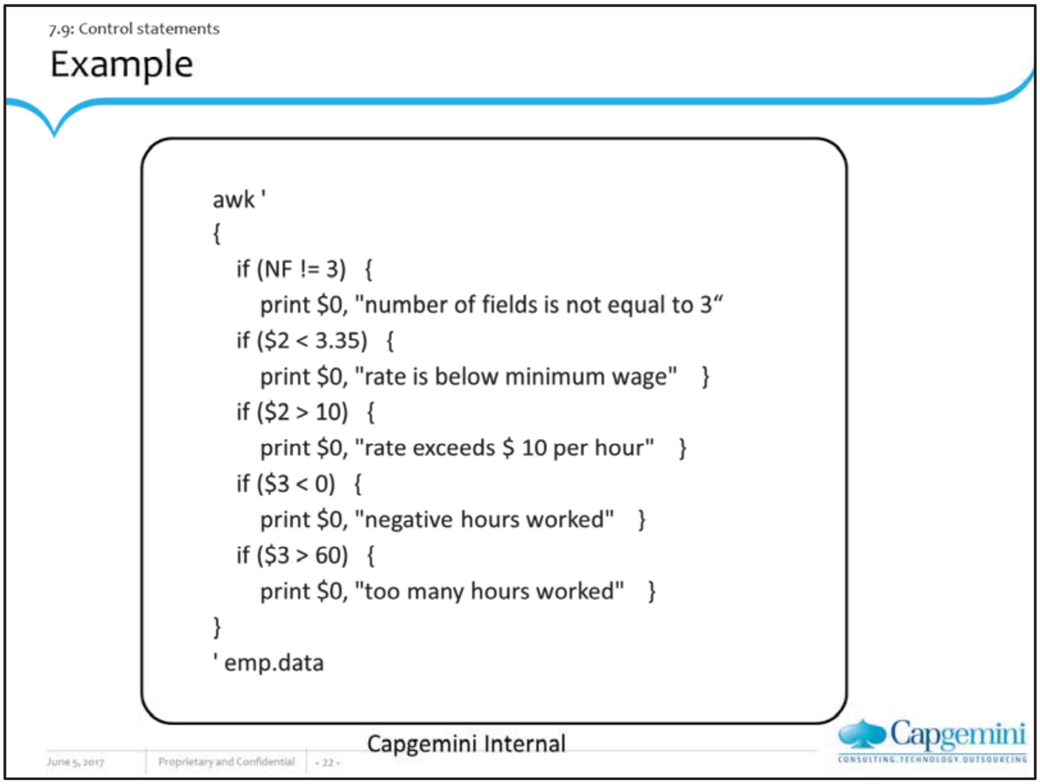
Control statements

```
if (<condition>)
{
                    <Block of statements>
}
else
{
                    <Block of statements>
}
```

```
While (Condition)
{
      <Block of statements>
}
for (<expr1>;<cond1>;<expr2>)
{
                    <Block of
statements>
}
```

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 21 -

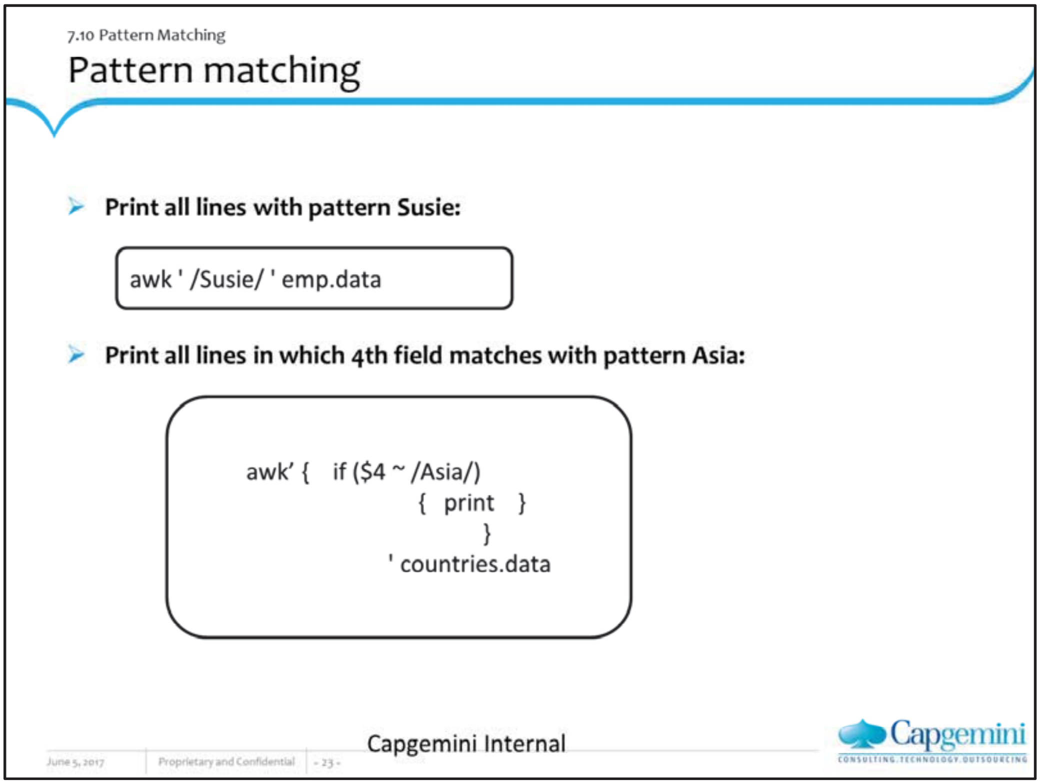


7.9: Control statements

Example

```
awk '
{
   if (NF != 3)   {
      print $0, "number of fields is not equal to 3“
   if ($2 < 3.35)   {
      print $0, "rate is below minimum wage"    }
   if ($2 > 10)   {
      print $0, "rate exceeds $ 10 per hour"    }
   if ($3 < 0)   {
      print $0, "negative hours worked"    }
   if ($3 > 60)   {
      print $0, "too many hours worked"    }
}
' emp.data
```

Capgemini Internal



7.10 Pattern Matching

Pattern matching

- Print all lines with pattern Susie:

```
awk ' /Susie/ ' emp.data
```

- Print all lines in which 4th field matches with pattern Asia:

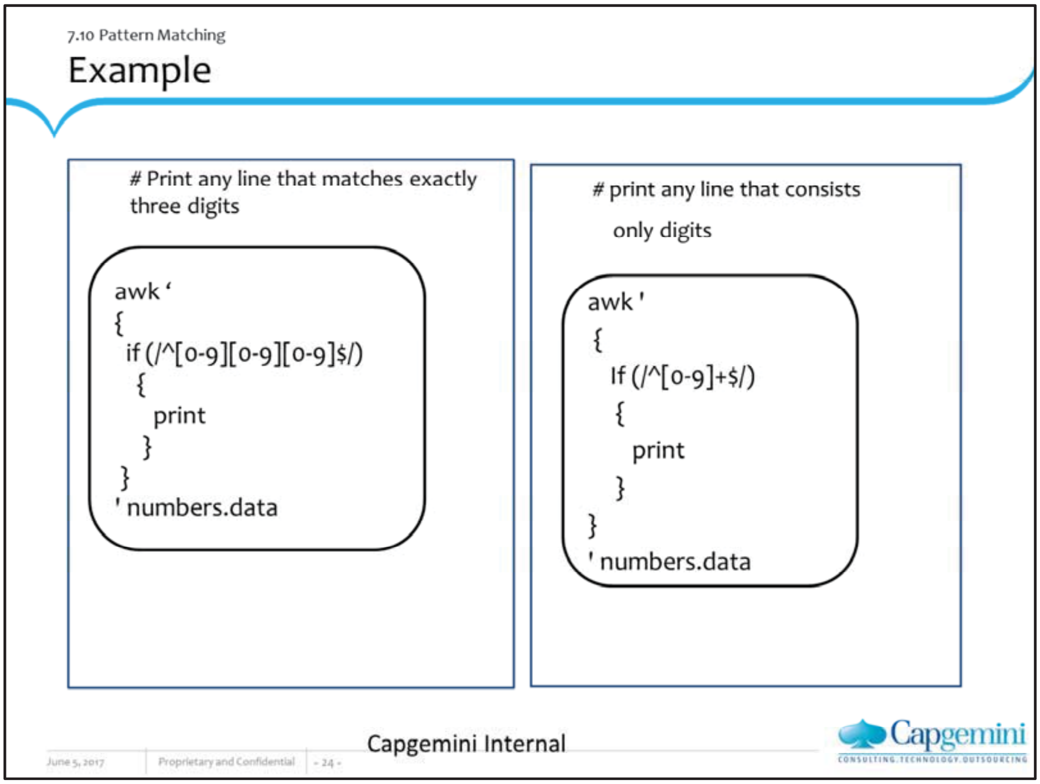
```
awk' {    if ($4 ~ /Asia/)
                  {  print   }
                        }
                ' countries.data
```

June 5, 2017 | Proprietary and Confidential | - 23 -

Capgemini Internal

To print all lines in which 4 th field does not matches with pattern Asia:

```
awk '
{
   if ($4 !~ /Asia/)
   {
       print
   }
}
' countries.data
```



7.10 Pattern Matching

Example

Print any line that matches exactly three digits

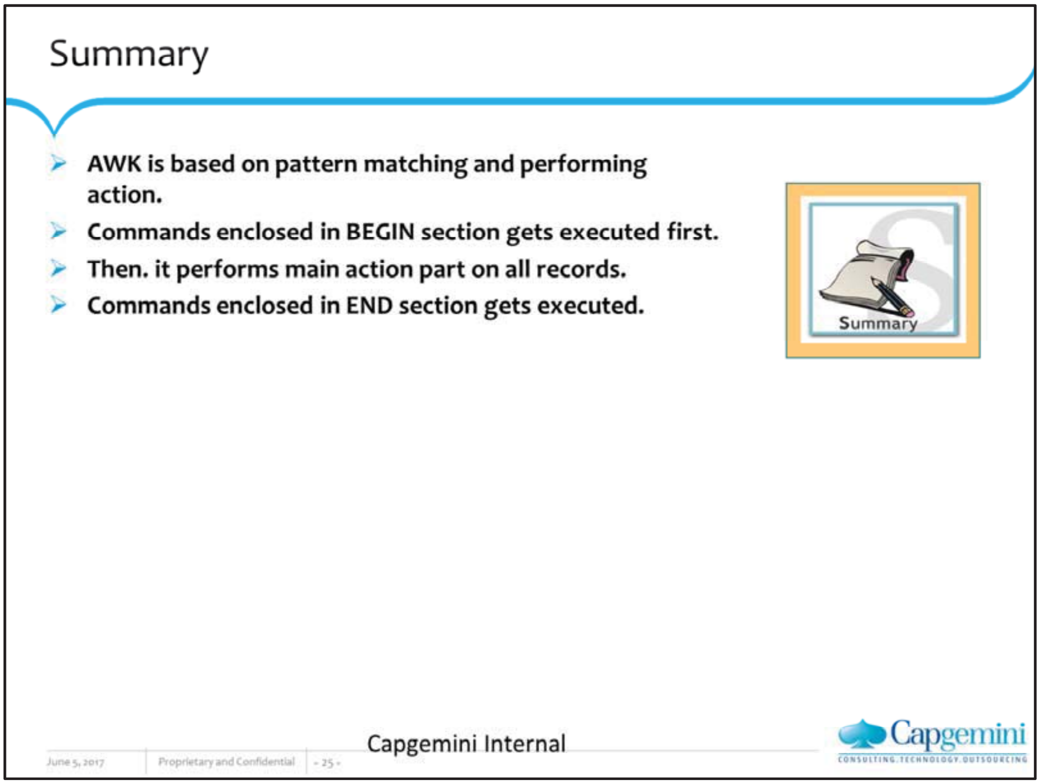
```
awk ‘
{
 if (/^[0-9][0-9][0-9]$/)
  {
    print
   }
 }
' numbers.data
```

print any line that consists only digits

```
awk '
 {
  If (/^[0-9]+$/)
  {
    print
  }
}
' numbers.data
```

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 24 -



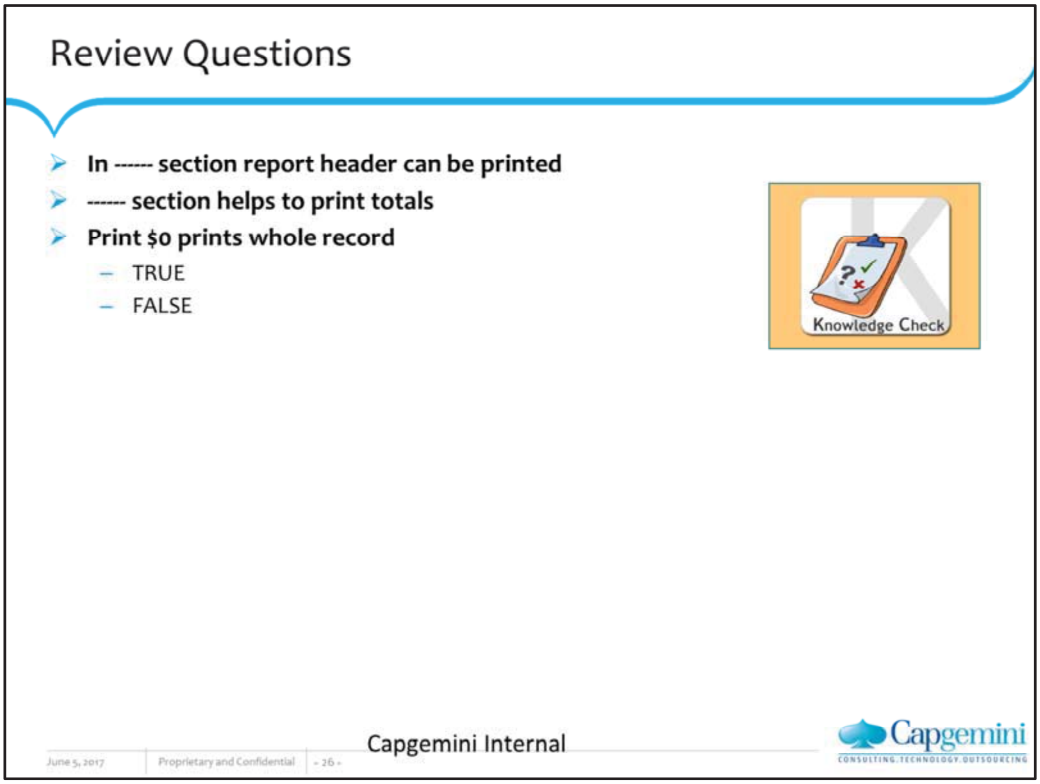
Summary

- AWK is based on pattern matching and performing action.
- Commands enclosed in BEGIN section gets executed first.
- Then. it performs main action part on all records.
- Commands enclosed in END section gets executed.

June 5, 2017 | Proprietary and Confidential | - 25 -

Capgemini Internal

Add the notes here.



Review Questions

- In ------ section report header can be printed
- ------ section helps to print totals
- Print $0 prints whole record
  - TRUE
  - FALSE

Knowledge Check

Capgemini Internal

June 5, 2017 | Proprietary and Confidential | - 26 -

Add the notes here.