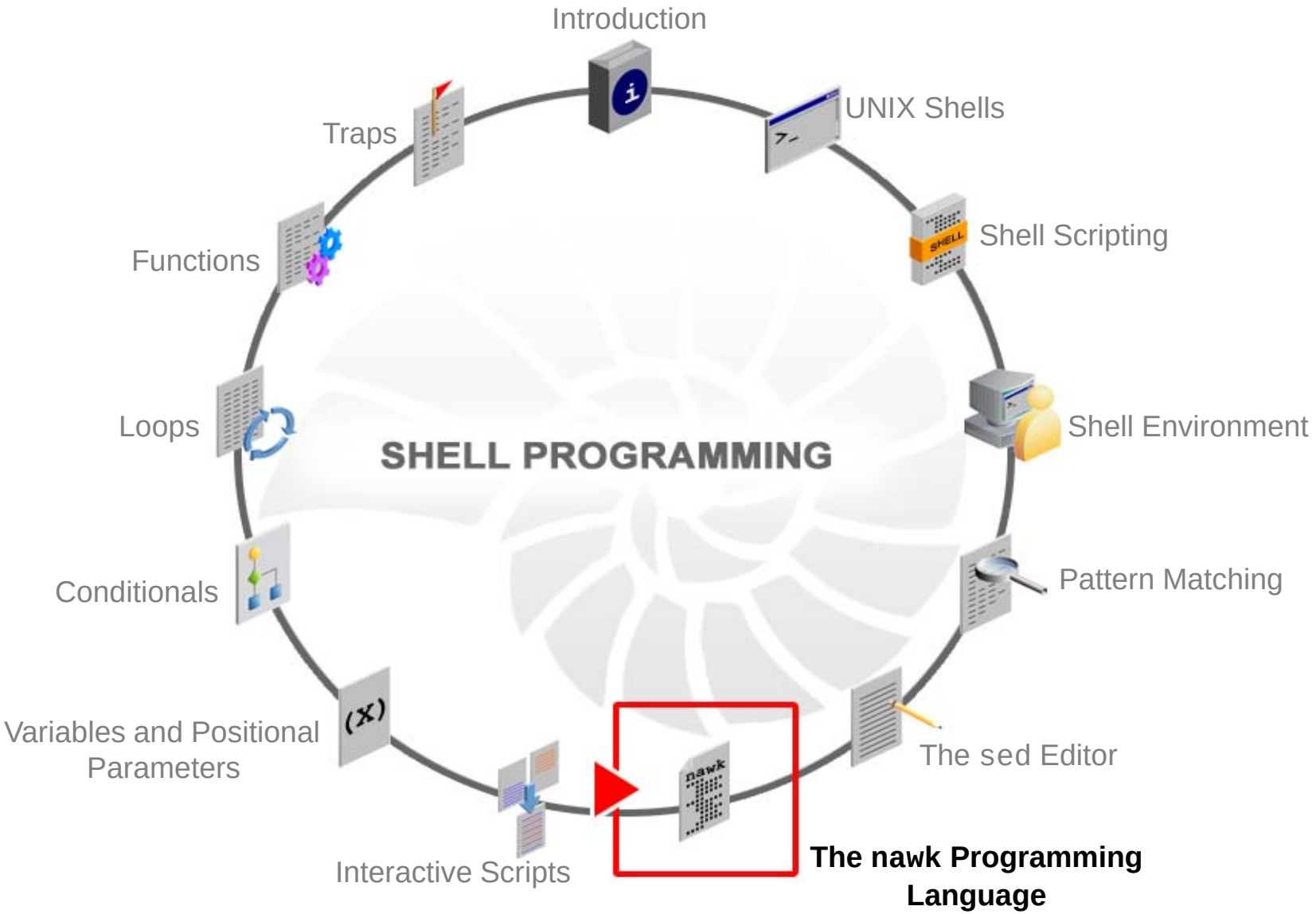
7

The nawk Programming Language



Objectives

After completing this lesson, you should be able to:

- Describe the capabilities of the `nawk` programming language
- Display output by using the `print` statement
- Perform pattern matching by using regular expressions
- Use the `nawk` built-in and user-defined variables

Agenda

- Describing the capabilities of the `nawk` programming language
- Displaying output by using the `print` statement
- Performing pattern matching by using regular expressions
- Using the `nawk` built-in and user-defined variables

awk Programming Language

- awk is named for its authors Aho, Weinberger, and Kernighan of AT&T Bell Labs.
- The awk programming language is:
  - A record-oriented language
  - A text processing, data extraction, and reporting tool
  - A language executed by the awk interpreter
- The language has three variations:
  - awk is the original and oldest version.
  - nawk (New awk) is the improved version adapted by most UNIX vendors.
  - gawk is the free GNU version.

nawk Programming Language

The nawk programming language:

- Looks at data by records and fields
- Uses regular expressions
- Uses numeric and text variables and functions
- Uses command-line arguments

nawk Capabilities

Applications written in the nawk programming language provide the following capabilities:

- Filtering
- Numerical processing on rows and columns of data
- Text processing to perform repetitive editing tasks
- Report generation

nawk Command Format

- Commands have the form:

```
nawk 'statement' input.file
```

- Scripts are executed with:

```
nawk -f scriptfile input.file
```

Agenda

- Describing the capabilities of the nawk programming language
- Displaying output by using the `print` statement
- Performing pattern matching by using regular expressions
- Using the nawk built-in and user-defined variables

Printing Selected Fields

- The `print` statement outputs data from the file.
- Command conventions:
  - Enclose the command in single quotation marks.
  - Enclose the command in braces `{}`.
  - Specify individual fields with `$1`, `$2`, `$3`, and so on.
  - Specify individual records with `$0`.

Printing Selected Fields

```
$ cat data.file
northwest   NW     Joel Craig     3.0 .98        3       4
western     WE     Sharon Kelly   5.3 .97        5       23
southwest   SW     Chris Foster   2.7 .8         2       18
southern    SO     May Chin       5.1 .95        4       15
southeast   SE     Derek Johnson  5.0 .70        4       17
eastern     EA     Susan Beal     4.4 .8         5       20
northeast   NE     TJ Nichols     5.1 .94        3       13
north       NO     Val Shultz     4.5 .89        5       9
central     CT     Sheri Watson   5.7 .94        5       13
$ nawk '{ print $3, $4, $2 }' data.file
Joel Craig NW
Sharon Kelly WE
Chris Foster SW
May Chin SO
Derek Johnson SE
Susan Beal EA
TJ Nichols NE
Val Shultz NO
Sheri Watson CT
```

Formatting with the `print` Statement

- Anything in double quotation marks is a string constant that can be used:
  - In `print` statements
  - As values to be assigned to variables among other things
- You can also use some special formatting characters:

Character	Octal Value	Meaning
\t	\011	Tab
\n	\012	Newline
	\007	Bell
	\042	“
	\044	$
	\045	%

Formatting with the `print` Statement

```
$ nawk '{ print $3, $4 "\t" $2 }' data.file
Joel Craig            NW
Sharon Kelly          WE
Chris Foster          SW
May Chin              SO
Derek Johnson         SE
Susan Beal            EA
TJ Nichols            NE
Val Shultz            NO
Sheri Watson          CT
```

Agenda

- Describing the capabilities of the `nawk` programming language
- Displaying output by using the `print` statement
- Performing pattern matching by using regular expressions
- Using the `nawk` built-in and user-defined variables

Using Regular Expressions

Regular expression metacharacters can be used in the pattern.

```
$ nawk '/east/' data.file
southeast  SE     Derek Johnson  5.0 .70        4       17
eastern    EA     Susan Beal     4.4 .8         5       20
northeast  NE     TJ Nichols     5.1 .94        3       13

$ nawk '/east/ { print $1, $5, $4 }' data.file
southeast 5.0 Johnson
eastern 4.4 Beal
northeast 5.1 Nichols

$ nawk '/east/ { print $1, $5 "\t" $4 }' data.file
southeast 5.0      Johnson
eastern 4.4        Beal
northeast 5.1      Nichols

$ nawk '/^east/' data.file
eastern    EA     Susan Beal     4.4 .8         5       20
```

Using Regular Expressions

```
$ nawk '/.9/' data.file
northwest   NW      Joel Craig     3.0 .98        3       4
western     WE      Sharon Kelly   5.3 .97        5       23
southern    SO      May Chin       5.1 .95        4       15
northeast   NE      TJ Nichols     5.1 .94        3       13
north       NO      Val Shultz     4.5 .89        5       9
central     CT      Sheri Watson   5.7 .94        5       13

$ nawk '/\.9/' data.file
northwest   NW      Joel Craig     3.0 .98        3       4
western     WE      Sharon Kelly   5.3 .97        5       23
southern    SO      May Chin       5.1 .95        4       15
northeast   NE      TJ Nichols     5.1 .94        3       13
central     CT      Sheri Watson   5.7 .94        5       13
```

The Special Patterns

There are two special patterns that are not used to match text in a file.

- `BEGIN`: An action to take before reading any lines
- `END`: An action to take after all lines are read and processed

The Special Patterns

```
$ nawk 'BEGIN { print "Eastern Regions\n" }; /east/ {
print $5, $4 }' data.file
Eastern Regions

5.0 Johnson
4.4 Beal
5.1 Nichol

$ nawk 'BEGIN {
> print "Eastern Regions\n"}; /east/ {print $5, $4}'
data.file
Eastern Regions

5.0 Johnson
4.4 Beal
5.1 Nichols
```

The Special Patterns

```
$ nawk 'BEGIN { print "Eastern Regions\n"}; /east/ {print
$5, $4}
> END {print "Eastern Region Monthly Report"}' data.file
Eastern Regions

5.0 Johnson
4.4 Beal
5.1 Nichols
Eastern Region Monthly Report
```

Using nawk Scripts

- A nawk script is a collection of nawk statements (patterns and actions) stored in a text file.
- To instruct nawk to read the script file, use the command:

```
nawk -f script_file data_file
```

Using nawk Scripts

```
$ cat report.nawk
BEGIN {print "Eastern Regions\n"}
/east/ {print $5, $4}
END {print "Eastern Region Monthly Report"}

$ nawk -f report.nawk data.file
Eastern Regions

5.0 Johnson
4.4 Beal
5.1 Nichols
Eastern Region Monthly Report
```

Using nawk Scripts

```
$ cat report2.nawk
BEGIN {print "** Acme Enterprises **"}
BEGIN {print "Eastern Regions\n"}
/east/ {print $5, $4}
END {print "Eastern Region Monthly Report"}

$ nawk -f report2.nawk data.file
** Acme Enterprises **
Eastern Regions

5.0 Johnson
4.4 Beal
5.1 Nichols
Eastern Region Monthly Report
```

Agenda

- Describing the capabilities of the `nawk` programming language
- Displaying output by using the `print` statement
- Performing pattern matching by using regular expressions
- Using the `nawk` built-in and user-defined variables

Built-in Variables

- As nawk processes an input file, it uses several variables.
- You can provide a value to some of these variables, whereas other variables are set by nawk and cannot be changed.
- A variable value can be a number, a string, or a set of values in an array.

Name	Default Value	Description
FS	Space or tab	The input field separator
OFS	Space	The output field separator
NR		The number of records from the beginning of the first input file

Input Field Separator (FS)

- The default input field separator (FS) is white space, which can be either a space or a tab.
- Frequently, other characters can separate the input, such as a colon or comma.
- You can set the input field separator variable with the `-F` option or set the value with an assignment.

```
nawk -F: 'statement' filename
nawk 'BEGIN { FS=":" } ; statement' filename
```

Input Field Separator (FS): Example

```
$ nawk 'BEGIN { FS=":" }; { print $1, $3 }' /etc/group
root 0
other 1
bin 2
sys 3
adm 4
uucp 5
mail 6
tty 7
lp 8
nuucp 9
staff 10
daemon 12
sysadmin 14
nobody 60001
noaccess 60002
nogroup 65534
```

Input Field Separator (FS): Example

```
$ cat report3.nawk
BEGIN { FS=":" }
{ print $1, $3 }
$ nawk -f report3.nawk /etc/group
root 0
other 1
bin 2
sys 3
adm 4
uucp 5
mail 6
tty 7
lp 8
nuucp 9
staff 10
daemon 12
sysadmin 14
nobody 60001
noaccess 60002
nogroup 65534
```

Output Field Separator (OFS)

- The default `OFS` is a space.
- In the `print` statement, a comma specifies using the `OFS`.
- If you omit the comma, the fields run together.
- You can also specify a field separator directly in the `print` statement, as in the following three lines:

```
$ nawk '{ print $3 $4 $2 }' data.file
$ nawk '{ print $3, $4, $2 }' data.file
$ nawk '{ print $3, $4 "\t" $2 }' data.file
```

Output Field Separator (OFS): Example

```
$ nawk 'BEGIN { OFS="\t" } ; { print $3, $4, $2 }'
data.file
Joel        Craig          NW
Sharon      Kelly          WE
Chris       Foster         SW
May         Chin           SO
Derek       Johnson        SE
Susan       Beal           EA
TJ          Nichols        NE
Val         Shultz         NO
Sheri       Watson         CT
```

Number of Records (NR)

- The number of records (NR) variable counts the number of input lines read from the beginning of the first input file.
- The variable's value is updated each time another input line is read.

Number of Records (NR): Example

```
$ more report4.nawk
{ print $3, $4, $2 }
END { print "The number of employee records is " NR }

$ nawk -f report4.nawk data.file
Joel Craig NW
Sharon Kelly WE
Chris Foster SW
May Chin SO
Derek Johnson SE
Susan Beal EA
TJ Nichols NE
Val Shultz NO
Sheri Watson CT
The number of employee records is 9
```

User-Defined Variables

- nawk allows you to create your own variables.
- Variable names should not conflict with the nawk variables or function names.
- The value assigned to a variable can be:
  - A string (string of characters)
  - A numeric
  - A literal value (”hello”)
  - The contents from a field from the input file

User-Defined Variables: Example

```
$ cat numexample.nawk
{ counter = counter + 1 }
{ print $0 }
END { print "*** The number of records is " counter }

$ nawk -f numexample.nawk data.file
northwest   NW     Joel Craig     3.0 .98        3       4
western     WE     Sharon Kelly   5.3 .97        5       23
southwest   SW     Chris Foster   2.7 .8         2       18
southern    SO     May Chin       5.1 .95        4       15
southeast   SE     Derek Johnson  5.0 .70        4       17
eastern     EA     Susan Beal     4.4 .8         5       20
northeast   NE     TJ Nichols     5.1 .94        3       13
north       NO     Val Shultz     4.5 .89        5       9
central     CT     Sheri Watson   5.7 .94        5       13
*** The number of records is 9
```

User-Defined Variables: Example

```
$ cat numexample2.nawk
{ total = total + $8 }
{ print "Field 8 = " $8 }
END { print "Total = " total }

$ nawk -f numexample2.nawk data.file
Field 8 = 4
Field 8 = 23
Field 8 = 18
Field 8 = 15
Field 8 = 17
Field 8 = 20
Field 8 = 13
Field 8 = 9
Field 8 = 13
Total = 132
```

User-Defined Variables: Example

```
$ cat numexample3.nawk
{ total = total + $8 }
{ print $0 }
END { print "The total of field 8 is " total }

$ nawk -f numexample3.nawk data.file
northwest   NW     Joel Craig     3.0 .98        3       4
western     WE     Sharon Kelly   5.3 .97        5       23
southwest   SW     Chris Foster   2.7 .8         2       18
southern    SO     May Chin       5.1 .95        4       15
southeast   SE     Derek Johnson  5.0 .70        4       17
eastern     EA     Susan Beal     4.4 .8         5       20
northeast   NE     TJ Nichols     5.1 .94        3       13
north       NO     Val Shultz     4.5 .89        5       9
central     CT     Sheri Watson   5.7 .94        5       13
The total of field 8 is 132
```

Additional Examples

```
$ nawk '/N[EOW] { print NR, $0 }' data.file
1 northwest        NW     Joel Craig     3.0 .98 3      4
7 northeast        NE     TJ Nichols     5.1 .94 3      13
8 north            NO     Val Shultz     4.5 .89 5      9

$ nawk 'BEGIN { count = 0 }
> /N/ { print NR, $0; count = count + 1 }
> END { print "count of North regions is", count }'
data.file
1 northwest        NW     Joel Craig     3.0 .98 3      4
7 northeast        NE     TJ Nichols     5.1 .94 3      13
8 north            NO     Val Shultz     4.5 .89 5      9
count of North regions is 3
```

Additional Examples

```
$ nawk '{ print "Record:", NR, $NF }' data.file
Record: 1 4
Record: 2 23
Record: 3 18
Record: 4 15
Record: 5 17
Record: 6 20
Record: 7 13
Record: 8 9
Record: 9 13
```

Additional Examples

```
$ nawk '{ print "Record:", NR, "has", NF, "fields." }' \
raggeddata.file
Record: 1 has 8 fields.
Record: 2 has 6 fields.
Record: 3 has 8 fields.
Record: 4 has 7 fields.
Record: 5 has 5 fields.
Record: 6 has 6 fields.
Record: 7 has 7 fields.
Record: 8 has 5 fields.
Record: 9 has 6 fields.
```

Additional Examples

```
$ nawk '{ print "Field 1 has", length($1), "letters." }'
raggeddata.file
Field 1 has 9 letters.
Field 1 has 2 letters.
Field 1 has 9 letters.
Field 1 has 8 letters.
Field 1 has 9 letters.
Field 1 has 7 letters.
Field 1 has 2 letters.
Field 1 has 3 letters.
Field 1 has 7 letters.
```

Writing Output to Files

In any statement that generates an output, it is possible to have that output redirected to a file name instead, as follows:

- Use the redirection symbol, >, to send data to a file.

```
$ nawk '{ print $2, $1 > "textfile" }' data.file
$ cat textfile
NW northwest
WE western
SW southwest
SO southern
SE southeast
EA eastern
NE northeast
NO north
CT central
```

- Use two redirection symbols, >>, to append to a file.

The `printf()` Statement

- The `printf()` statement allows you to print a character string.
- This string can contain place holders that represent other values.
- These place holders can represent integers, floating points, characters, and character strings.
- Syntax:

```
printf ("string_of_characters" [ , data_values ] )
```

Note: For every place holder in the character string of `printf()`, there must be a value following the character string. These values must be separated by a comma.

The `printf()` Statement

```
$ nawk '{ printf "%10s %3d \n", $4, $7 }' data.file
Craig       3
Kelly       5
Foster      2
Chin        4
Johnson     4
Beal        5
Nichols     3
Shultz      5
Watson      5
```

Note: The `printf` statement does not use the `OFS` variable because of the `printf` statement's inherent ability to format the output line. So adjusting the fields to appear in column format is done by preceding the format specification with a digit, which refers to the number of spaces that the value should consume.

Summary

In this lesson, you should have learned how to:

- Describe the capabilities of the `nawk` programming language
- Display output by using the `print` statement
- Perform pattern matching by using regular expressions
- Use the `nawk` built-in and user-defined variables

Practice 7 Overview: The nawk Programming Language

This practice covers the following topics:

- Using nawk and Regular Expressions
- Using nawk to Create a Report