

CS-333 - Unix Operating System Fundamentals

Lecture 12 awk

Based on slides created by Dr. Bangalore for the Spring 2005 offering of the course

awk (Aho, Weinberger, Kernighan)

- A pattern matching tool
- Perform specified operations on specific fields in records after a pattern is matched
- Fully programmable (conditional expressions, loops, and variables)
- Reads data files or input from other utilities
- More info on www.gnu.org/software/gawk/manual/

11/17/2006

3

Usage

awk [option] '/pattern/ {action}' filename

- Examines each line in the input file for a match with the specified pattern
- The action is performed on the lines that match the pattern
- The input file is not modified
- The default action is to print the line that matches the selection criteria
- awk requires the file to be structured by fields

11/17/2006

4

awk variables

- Field variables (predefined)
 - Hold the values of the fields on each input line e.g., \$1, \$2, ...
 - *awk '/pattern/ {print \$1 \$3 \$2}' filename*
 - \$0 is the entire line
- User variables
 - use the option -v
 - *awk -v variable='value' '{action}' filename*
- Strings are interpreted as variables unless they are inside "" or //
- Numbers are not quoted and are not interpreted as variables

11/17/2006

5

awk's capabilities (1)

- Default delimiters are space and tab
- Use -F to use other characters as delimiters
 - *awk -F: '/pattern/ {action}' filename*
- Regular expressions can be used to specified the patterns:
 - [], ^, \$, *, ...

11/17/2006

6

Department of Computer and Information Sciences UAB

awk's capabilities (2)

- Selecting lines by field value
 - `awk '$5 == "y" {print $0}' filename`
 - `awk '$3 == 3.79' filename`
 - `awk '$3 < 3.79' filename`
- Logical operators
 - `awk '/pattern1/||/pattern2/ {action}' filename`
 - `awk '$3 < 4.00 && $3 > 2.00' filename`
 - `awk '$3 < 4.00 || $3 > 2.00' filename`

11/17/2006 7

Department of Computer and Information Sciences UAB

awk's capabilities (3)

- More regular expressions (~ stands for 'contains')
 - `awk '$3 ~ /\.89/' filename`
 - `awk '$3 ~ /ry/' filename`
- Logical negation
 - `awk '!($3 == "home")' filename`
 - `awk '$3 !~ /pattern/ {action}' filename`
- Number of fields (NF)
 - NF is a variable holding the number of fields in an input line
 - `awk '! (NF == 4)' filename`

11/17/2006 8

Department of Computer and Information Sciences UAB

awk's command files (1)

- Command files are not executable files, but text files that are read by awk
- Contain pattern-action statements
 - `awk -f command.file filename`
- Selecting lines based on the line number
 - NR (Number of Record) is a predefined variable

```
$ cat findNR
NR == 6 {print}

$ awk -f findNR filename
field1 field2 ...
```

11/17/2006 9

Department of Computer and Information Sciences UAB

awk's command files (2)

- Specifying the input field separator in a command file
 - FS (Field Separator) is a predefined variable

```
$ cat field-separator-in
BEGIN { FS=";" }
{print $1, $2}
```
- Specifying the output field separator in a command file
 - OFS (Output Field Separator) is a predefined variable

```
$ cat field-separator-out
BEGIN {
  FS=";"
  OFS="|"
}
{print "Fields: " $1, $2}
```

11/17/2006 10

Department of Computer and Information Sciences UAB

awk's command files (3)

- Specifying the output record separator in a command file
 - ORS (Output Record Separator) is a predefined variable

```
$ cat record-separator-out
BEGIN {
  FS=";"
  OFS="+"
  ORS="-----"
}
{print "NR is " NR, $1, $2}
```
- Specifying the input record separator in a command file
 - RS (Record Separator) is a predefined variable

```
$ cat record-separator-in
BEGIN {
  RS=";"
  FS=";"
}
{print "NR is " NR, $1, $2}
```

11/17/2006 11

Department of Computer and Information Sciences UAB

More on variables ...

- Use variables to improve readability
 - Instead of `/pattern/ {print $1, $2}`
 - Use


```
/pattern/ {
  var1 = $1
  var2 = $2
  print var1, var2
}
```

11/17/2006 12

Arithmetic operations

- Addition and subtraction
 - { *print \$1, \$2 + 0.5* }
 - { *print \$1, \$2 - 0.5* }
- Multiplication and division using variables

```
{
    price = $1
    quantity = $2
    subtotal = price * quantity
    tax = 0.1
    print "Total = " subtotal + (subtotal * tax)
}
```

- Arithmetic operations are floating-point
- +=, -=, *=, /= are supported

11/17/2006

13

printf

- Formatting capabilities over print

```
$1 == "yes" {
    price = $2
    tax = 0.09
    total = price + price * tax
    printf "Item: %s Total: %s\n", $2, total
}
```

- Left and right justification
 - *printf "%-20s %10s\n", \$1, total*
- Decimal alignment and truncating numbers
 - *printf "%-20s %10.2f\n", \$1, total*

11/17/2006

14

BEGIN & END patterns

- Used to perform task before or after awk's main loop
 - receive input → operate on input → terminate

```
BEGIN {
    printf "The name and price for each item is:\n"
}
{
    name = $1
    price = $3
    quantity = $4
    total = price * quantity
    sum += total
    printf "%-12s %5.2f\n", name, total
}
END {
    printf "\nThe total cost of all items is: %.2f\n", sum
}
```

11/17/2006

15

Much more on awk...

- Support for all C-like statements (*if, while, for, break,...*)
- There are many books written on awk...
- Interesting trivia: awk is the only scripting language beside the Bourne shell that is available in standard Unix environment
- Implementations of awk as installed software are available for almost all other operating systems

11/17/2006

16