AWK with a Text File Table

Regular Wizardry

Text files formatted as tables are easily searched and modified using AWK. Admins in particular appreciate AWK, which is typically installed on any flavor of Unix.

BY NICO HOCHGESCHWENDER

Automatic text file manipulation is part of any admin's daily grind. This important task may involve evaluating logfiles, creating or modifying configurations, or adding new accounts.

For some tasks, the classic Unix programming language AWK offers the most efficient solution. AWK is a compact language with syntax similar to C, which makes AWK easy to learn for anyone with C experience.

An AWK script parses the input file line by line, searching for patterns. When AWK finds a match, it performs a specified action. If the programmer does not specify a pattern, AWK simply performs the action for each line. As you'll learn in this article, AWK is a very efficient tool for searching any kind of text file, including a previously prepared table stored in text format. A formatted text file, accompanied by a few simple AWK commands or scripts, can serve as a very simple and flexible data retrieval system without the complication or expense of an SQL database. This article describes how a system administrator can use AWK to obtain information about computers on a local network.

The free (i.e., released under the GPL) AWK version *gawk* [1] is a standard component of any Linux distribution. And as traditional Unix systems also include AWK, the tool is particularly useful for platform-independent scripting. If you have Solaris, HP/UX and AIX servers as well as Linux computers, AWK could become an indispensable tool.

The examples in this article are based on the list of computers in Listing 1. The table shows the contents of a text file that lists the computer name, IP address, operating system, software, and RAM for each computer. The idea is that, when a computer is reconfigured or added to the network, the system administrator updates the information in this file.

Approach

The basic syntax for a gawk single-line script is as follows:

```
gawk [options]↩
'<I(>AWK program<I>'↩
 <I>input_file<I>
```

Larger AWK scripts should be stored in a file. In that case, the syntax is *gawk [options] -f scriptfile inputfiles*. The first thing we would like AWK to do now is give us a list of the computer names in our sample file (Listing 1):

```
gawk '{print $1}' list
```

The field *$1* refers to the first column. If you need to know the IP address instead, simply replace *$1* with *$2*. *$0* corresponds to the full line. *gawk* '{*print $0*}' *list* outputs the complete file on the screen.

Examples

The following search key will give us the whole range of information for a computer called Goofy1:

```
gawk '$1=="Goofy1" {print $0}'↩
 list
```

In each line, AWK checks if the expression *Goofy1* occurs exactly once in column *$1*. If so, it prints the whole line, {*print $0*}. Instead of the equal to operator (==), this example uses the negation operator, *!=*, to ensure that AWK will only run the command if there is no match (see Table 1.) The *gawk '$1 != "Goofy1" {print $0}' list* command sends the contents of lines without the *Goofy1* string to standard output.

AWK can search for ranges in addition to individual search patterns. The following syntax uses two regular expressions (see Table 2), surrounded by slash characters; AWK compares them with the whole line.

```
gawk '/Goofy1/,↩
/Asterix/ ↩
{print $0}' list
```

The output is the whole area from the *Goofy1* search string up to and including *Asterix1*.

Logical Operators

Search keys can be extended and linked using Boolean operators (Table 1) such as the AND operator.

```
gawk '($3=="OSX") && ↩
($4=="Photoshop") ↩
  {print $1}' list
```

Logical operands need to be put in round brackets, and this can cause errors, especially if you need to deal with more complex expressions. AWK has a logical OR operator, || in addition to the logical AND.

Output

Thus far we have been happy to send output to the screen. But just like the shell, AWK is capable of redirecting data streams into files:

```
gawk '$1=="Obelix" ↩
{print $0 >   "/home/↩
linux/test"}' list
```

The preceding command sends the data stream to a file called *test*. If the file does not exist, AWK will automatically create it. You can redirect output using > >. It is okay to use the shell's own redirection function for this simple example, but the AWK variant allows you to redirect output to different files and view the output on the screen at the same time.

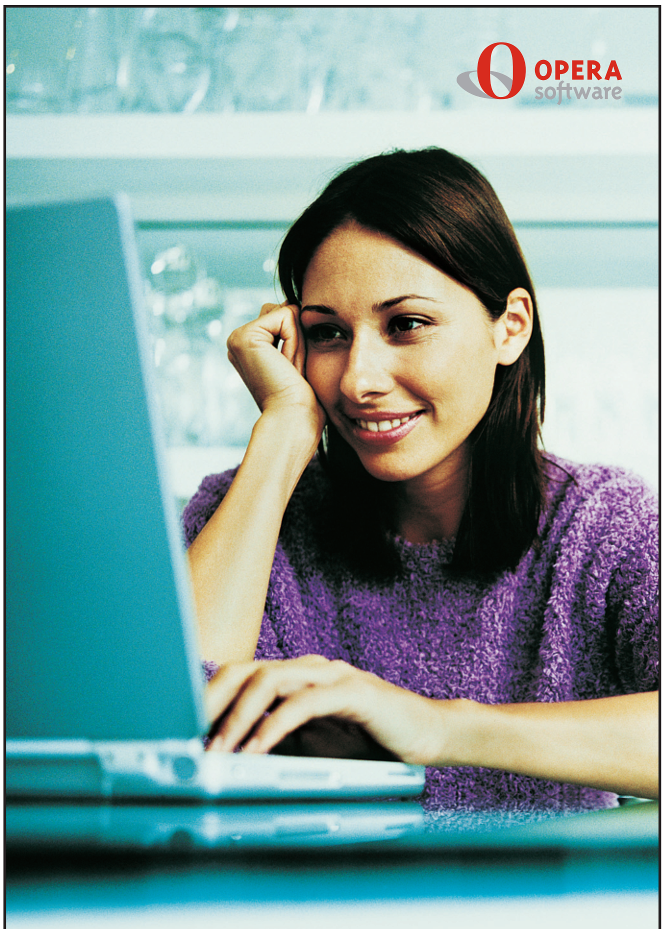
AWK also understands the *printf()* function, which is also used by C and the shell. Admins can use *printf()* for enhanced output formatting. Just like its counterpart in C, *printf()* does not wrap the output but expects the programmer to add \n for new lines.

```
gawk '{printf↩
("%x\n",$5)}' list
```

In this example, we want *printf* to output an integer

Listing 1: Computer list

```
DagobertDuck      10.1.1.3    Debian   Kylix         256
Goofy1            10.1.1.4    Solaris  Mathematica   512
MickeyMouse       10.1.1.5    Debian   Apache        512
LuckyLuke1        10.1.1.6    Debian   Samba         256
LuckyLuke2        10.1.1.7    Debian   Eclipse       256
LuckyLuke3        10.1.1.8    Suse     Mupad         256
LuckyLuke4        10.1.1.9    Debian   Mupad         128
LuckyLuke43       10.1.1.10   Debian   Mupad         128
LuckyMickeyMouse   10.1.1.1    Debian   Mupad
128
Asterix1          10.1.1.12   RedHat   NetBeans      128
Asterix2          10.1.1.13   Debian   NFS           256
Obelix            10.1.1.14   RedHat   ICC           256
Apple1            10.1.1.15   OSX      Photoshop
1024
Apple2            10.1.1.16   OS6      Photoshop     128
Apple3            10.1.1.17   OSX      Photoshop     512
```



Opera 7 for Linux

Take control of your Web experience.

A eb standards compliant and full-featured Internet power tool, Opera includes custom pop-up blocking, tabbed browsing, integrated searches, advanced security management, and unique functions like Opera's groundbreaking e-mail client, RSS Newsfeeds, and IRC chat. And because we know that people have different needs, you can customize the look, feel, and features of the Opera browser with a few clicks of the mouse.

It's your Web. Enjoy.

**ownloa Opera from
www.opera.com/linuxmag**

value in hex (*%x*) and then add a new line (*\n*). The argument passed to gawk is the content of column five. Refer to [2] for more detail on this.

A New Start

The *BEGIN* and *END* constructs are useful for outputting headlines or messages. AWK runs any *BEGIN* commands before parsing the input file and any *END* commands after completing the last line.

```
gawk 'BEGIN ↵
{print "Search for MickeyMouse"}
$1=="MickeyMouse" {print $0}
END {print "-------"}' list
```

Besides string manipulation, AWK can also handle numerical operations. The last line in the file in Listing 1 contains numbers, which AWK can manipulate numerically or non-numerically. For example, you can type the following command if you need to know how much memory you have in your lab environment:

```
gawk '{sum+=$5; print $5, sum}'↵
 list
```

This mini-program adds field five in every line and stores the current total in a variable called *sum*. It outputs the value of the field and the sum.

Table 1: AWK Operators

Operator	Explanation
$	Field operator
++−	Postfix increment and decrement
++−	Prefix increment and decrement
^	Power
!	Logical negation
+ -	Sign operations
* / %	Multiplication, division, modulo operation
+ -	Addition, subtraction
<	Less than
<=	Less than or equal to
==	Equal to
!=	Unequal to
>=	Greater than or equal to
>	Greater than
~ !~	Compare to regular expression
&&	Logical AND
||	Logical OR
=	Assignment
+=	Addition and assignment
-=	Subtraction and assignment
*=	Multiplication and assignment
/=	Division and assignment
%=	Modulo operation and assignment
^=	Power and assignment

Regular Expressions

Regular expressions are often useful if you need to manipulate or search text documents. Meta-characters give you the ability to create quite complex search keys. AWK supports regular expressions:

```
gawk '$1 ~ /[0-9]/
 {print $0}'↵
 list
```

This script searches column one, *$1*, for a search key that contains a line number between 0 and 9. To tell AWK only to search in *$1*, you need to explicitly assign the column number to the search key using a tilde (*~*) character. The negation operator would achieve exactly the opposite effect: *!~* searches any lines in which the regular expression does not occur.

To find any computers whose names end in *Duck* in the list, we need the following command:

```
gawk '$1 ~ /Duck$/ {print $0}'↵
 list
```

The dollar operator in */Duck$/* appends the regular expression to the end of field *$1*. */^Lucky/* finds any entries that start with *Lucky*, such as *LuckyLuke* or *LuckyMickeyMouse*. Boolean operators provide a useful extension to this functionality:

```
gawk '$1 ~/(y|M)/ {print $0}'↵
 list
```

This command searches the first column for occurrences of *y* or *M*. Table 2 gives you an overview of meta-characters.

Table 2: Regular expressions

Expression	Explanation
.	Replaces an arbitrary character
^	Finds the following regular expression at the start of the line
$	Finds the following regular expression at the end of the line
[ ]	Finds any character between the square brackets
[a-d1-7]	Character classes with ranges: all letters between a and d, and all numbers between 1 and 7
X?	Either no Xs or exactly one X
X*	Either no Xs or more than one X
X|Z	X or Z
XZ	X immediately followed by Z

String Functions

AWK has a wide range of functions for string replacement or substitution. In our sample file, only one computer has a Suse operating system. Let's assume that the administrator who manages this network migrates this computer to Debian and now needs to update the computer list. Instead of using an editor, the admin could do the job more elegantly using an AWK string function.

```
gawk ↵
'{sub(/Suse/, "Debian", $3); ↵
 print >> "/home/linux/test"}' ↵
 list
```

The preceding command passes the search key, */Suse/*, the replacement text *"Debian"*, and the column *$3*, to the *sub()* (substitute) function. Assuming that the search key occurs in this column, the replacement text is substituted in. If you need more information on string functions, check out [3] and the manpage.

Going Bigger

More complex AWK scripts allow you to define your own functions, loops, and multi-dimensional arrays. The GNU variant can even handle TCP/IP communications [4]. ■

INFO

[1] GNU AWK: *http://www.gnu.org/software/gawk/*

[2] Printf examples from the gawk manual: *http://www.gnu.org/software/gawk/manual/html_node/Printf-Examples.html*

[3] Helmut Herold, "AWK and SED": Addison Wesley, 1991

[4] TCP/IP communication with gawk: *http://www.gnu.org/software/gawk/manual/html_node/TCP_002fIP-Networking.html*

THE AUTHOR

Nico Hochgeschwender is studying Computer Science and majoring in mobile robotics. When he has time to spare, he enjoys mountaineering or cycle racing.