Get started with GAWK: AWK language fundamentals

Begin learning AWK with the open source GAWK implementation

Skill Level: Intermediate

Michael Stutz (stutz@dsl.org)
Author
Consultant

19 Sep 2006

Discover the basic concepts of the AWK text-processing and pattern-scanning language. This tutorial gets you started programming in AWK: You'll learn how AWK reads and sorts its input data, run AWK programs, manipulate data, and perform complex pattern matching. When you're finished, you'll also understand GNU AWK (GAWK).

Section 1. Before you start

Learn what to expect from this tutorial, how to get the most out of it, and what you need to work through it.

About this tutorial

GNU AWK (GAWK) is an open source implementation of the venerable AWK programming language and it is available for all UNIX® systems. The AWK language is a UNIX standby -- a powerful text-manipulation and pattern-matching language that is particularly suited for information retrieval, making it ideal for use with today's database-driven applications. Because of its integration with the UNIX environment, complete working programs can be conceived, built, and executed quickly, with immediate results.

This tutorial provides a hands-on introduction to the AWK text-processing language.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

It shows how to use the open source GAWK interpreter to write and execute AWK programs so that you can search and manipulate your data in various ways.

Objectives

This tutorial is written for anyone who wants to begin harnessing the text-manipulation power of AWK. In this tutorial, you use GAWK to learn the various methods of running AWK programs. You also see how programs are structured and learn AWK's record and field paradigm. By the time you have completed the tutorial, you will have learned the rudimentary aspects of the language, including formatting output, record and field manipulation, and pattern matching. You should also be able to write custom AWK programs to perform complex text processing from the UNIX command line.

Prerequisites

This tutorial has no particular prerequisites, although you should be familiar with using a UNIX command-line shell. Some familiarity with the constructs of the C programming language is helpful, but it is not required.

System requirements

You must have a working copy of GAWK installed on your system, preferably Version 3.0 or later. GAWK is easily obtainable in both source and binary packages (see Resources). If you're installing GAWK from source, consult the README file in the GAWK source distribution, which lists any additional software requirements for successful compilation and installation.

Section 2. Get ready to GAWK

Learn about the AWK programming language and the differences between implementations, and prepare your GAWK installation so that you can begin programming.

The AWK language

AWK is the name of the programming language itself, written in 1977. Its name is an acronym for the surnames of its three principal authors: Drs. A. Aho, P. Weinberger, and B. Kernighan.

Because AWK is a text-processing and pattern-matching language, it's often called a

© Copyright IBM Corporation 1994, 2008. All rights reserved.

data-driven language -- the program statements describe the input data to match and process rather than a sequence of program steps, as is the case with many languages. An AWK program searches its input for records containing patterns, performing specified actions on that record until the program reaches the end of input. AWK programs are excellent for work on databases and tabular data, such as for pulling out columns from multiple data sets, making reports, or analyzing data. In fact, AWK is useful for writing short, one-off programs to perform some feat of text hackery that in another language might be overkill. In addition, AWK is often used on the command line or with pipelines as a *power tool*.

Like Perl -- which it inspired -- AWK is an interpreted language, so AWK programs are generally not compiled. Instead, the program scripts are passed to the AWK interpreter at run time.

Systems programmers find themselves immediately at home with the C-like syntax of AWK's input language. In fact, many of its features, including control statements and string functions, such as `printf` and `sprintf`, appear virtually identical. However, some differences do exist.

Versions of AWK

The AWK language was updated and more or less replaced in the mid-1980s with an enhanced version called NAWK (*New AWK*). The old AWK interpreter still exists on many systems, but it's often installed as the `oawk` (*Old AWK*) command, while the NAWK interpreter is installed as the main `awk` command as well as being available as `nawk`. Dr. Kernighan still maintains NAWK; like GAWK, it is open source and freely available (see Resources).

GAWK is the GNU Project's open source implementation of the AWK interpreter. While the early GAWK releases were replacements for the old AWK, it has since been updated to contain the features of NAWK.

In this tutorial, *AWK* always refers to references general to the language, while those features specific to the GAWK or NAWK implementations are referred to by their names. You'll find links to GAWK, NAWK, and other important AWK sites in the Resources section.

GAWK features and benefits

GAWK has the following unique features and benefits:

- It is available for all major UNIX platforms as well as other operating systems, including Mac OS X and Microsoft® Windows®.
- It is Portable Operating System Interface (POSIX) compliant and contains all features from the 1992 POSIX standard.
- It has no predefined memory limits.
- Helpful new built-in functions and variables are available.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- It contains special `regexp` operators.
- Record separators can contain `regexp` operators.
- Special file support is available to access standard UNIX streams.
- Lint checking is available.
- It uses extended regular expressions by default.
- It allows unlimited line lengths and continuations with the backslash character (\).
- It has better, more informative error messages.
- It includes TCP/IP networking functions.

Check your version

After you have installed GAWK, you must first determine where your local copy has been placed. Most systems use GAWK as their primary AWK install, such as /usr/bin/awk as a symbolic link to /usr/bin/gawk, so that `awk` is the name of the command for the GAWK interpreter. This tutorial assumes such an installation. On systems with another flavor of AWK already installed or taking precedence, you might have to call GAWK as `gawk`.

You'll know that you have everything installed correctly if you type `awk` and get the GNU usage screen, as shown in Listing 1. Most other flavors of AWK return nothing at all.

Listing 1. GAWK installed as awk

```
$ awk
Usage: gawk [POSIX or GNU style options] -f progfile [--] file ...
Usage: gawk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options:
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
        -v var=val              --assign=var=val
        -m[fr] val
        -W compat               --compat
        -W copyleft             --copyleft
        -W copyright            --copyright
        -W dump-variables[=file]        --dump-variables[=file]
        -W gen-po               --gen-po
        -W help                 --help
        -W lint[=fatal]         --lint[=fatal]
        -W lint-old             --lint-old
        -W non-decimal-data     --non-decimal-data
        -W profile[=file]       --profile[=file]
        -W posix                --posix
        -W re-interval          --re-interval
        -W source=program-text  --source=program-text
        -W traditional          --traditional
        -W usage                --usage
        -W version              --version

To report bugs, see node `Bugs' in `gawk.info', which is
section `Reporting Problems and Bugs' in the printed version.
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

As you can see, GAWK takes the GNU-standard option for getting the version. The output you get, including a notice from the Free Software Foundation concerning the licensing of GAWK and its lack of warranty, should look like Listing 2.

Listing 2. Displaying the GAWK version

```
$ gawk --version
GNU Awk 3.1.5
Copyright (C) 1989, 1991-2005 Free Software Foundation.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
$
```

Now that you have a working GAWK installation and you know how to call it, you're ready to begin programming. The next section describes basic AWK programming concepts.

Section 3. Understanding records, fields, and rules

Learn the basics of the AWK programming language: records, fields, and rules.

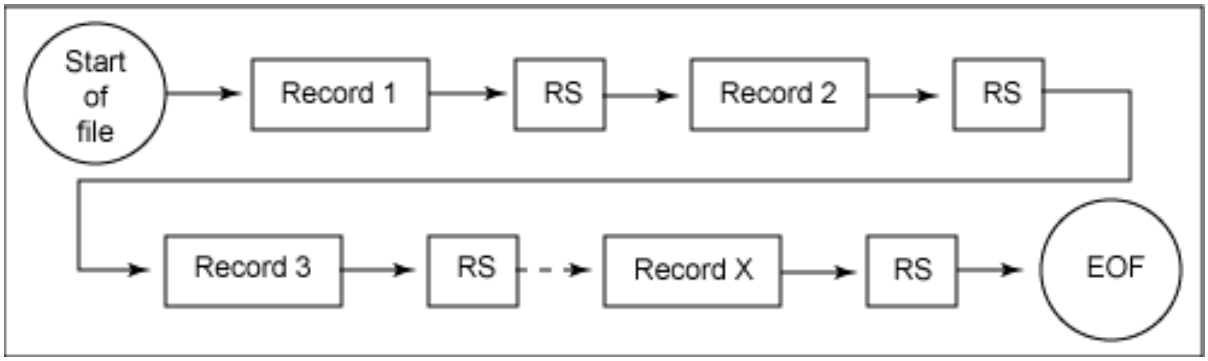
Elements of an AWK input file

AWK works on text input -- and this text can be a file or the standard input stream -- which it sorts into records and fields. An AWK *record* is a single, continuous length of input data on which AWK works. Records are bounded by a *record separator,* which is a character string and defined as the RS variable (see Change the record separator). By default, the value of RS is set to a newline character, so the whole lines of input are considered records for the default behavior of AWK.

Records are read continuously until the end of the input is reached. Figure 1 shows how input data is broken into records.

Figure 1. AWK input data split into records

© Copyright IBM Corporation 1994, 2008. All rights reserved.

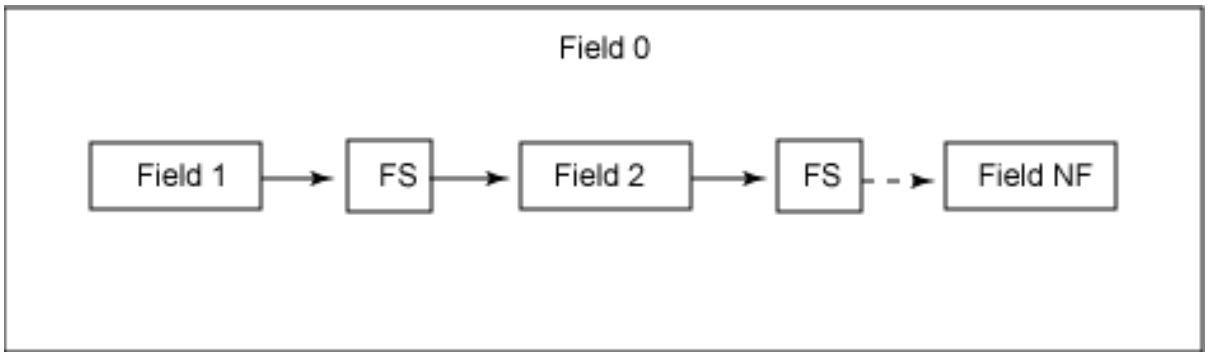


Each record, in turn, can be broken down further into individual chunks called *fields.* Like records, fields are bounded. The default *field separator* is any amount of whitespace, including tab and space characters. So by default, lines of input are further broken down into individual words (any groups of characters separated by whitespace).

You can reference the fields of a record by field number, beginning with 1. You can call the last field in each record either by its number or with the special variable `NF`, which contains the number of fields in the current record.

You can reference an entire record, including all of its fields and field separators, by a field number of 0. Figure 2 shows the contents of such a field -- a single record of AWK input data as it is divided into its composite fields.

Figure 2. Fields of an AWK record



Note: AWK provides ways to split the elements of a field into its character components, but that's outside the scope of this tutorial.

Know the rules

AWK programs consist of *rules,* which are patterns followed by actions separated by newlines. When AWK follows a rule, it searches the input records for matches to the given pattern, and then performs the given action on those records:

```
/pattern/ { action }
```

You can omit either the pattern or the action in a rule.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Actions consist of AWK statements and are separated by semicolons (;). You can give multiple rules on the same line, too -- but then you must separate them with semicolons, as well. When a rule only contains an action, it's performed on every record in the input. When a rule only contains a pattern, the records that match those patterns are printed to the output.

The empty pattern, `//`, matches the null character, and it is the same as giving a rule with no pattern at all. However, when the empty action is given, `{}`, it's not the same as giving no action at all. An empty action specifies doing nothing (and therefore, the record is not printed).

Print each record of the input

You can write a rule to print each and every record of input in a few ways, most simply as:

```
//
```

This is effectively the same as:

```
{ print }
```

Formally, you could do:

```
// { print }
```

This is about the simplest program in AWK. While it doesn't do anything spectacular, programs not much bigger than this can perform very complex operations.

Use the BEGIN and END patterns

AWK contains two special patterns: `BEGIN` and `END`. Both are given without slashes. The `BEGIN` pattern specifies actions to be performed before any records are processed:

```
BEGIN {action}
```

The `END` pattern specifies actions to be performed after all records are processed:

```
END {action}
```

`BEGIN` is frequently used to set variables; `END` is often used to tabulate, total, and process data and figures that were read in the various input records. Together, these patterns are often used to output text before and after the processed input text is output.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

For example, you can output messages when the program begins and right before it ends, as in the AWK program given in Listing 3.

Listing 3. A program with BEGIN and END patterns

```
# Program to print a file with a header and footer.

BEGIN { print "Beginning of file";
    print "-----------------" }

// # Print every line in the file.

END { print "------------";
      print "End of file."}
```

Notice that statements are separated by semicolons and that program comments begin with a hash character.

Some essential actions

You typically build actions with AWK language statements, such as the `print` statement in Listing 3. Table 1 lists some of the most common GAWK statements and functions used in actions and describes their meanings. For `C` programmers, note the similarities between the statements and functions of the language and the AWK equivalents.

Table 1. Common GAWK statements

Statement	Description
`exit`	Stops the execution of the program and exits.
`next`	Stops processing the current record and immediately advances to the next record.
`nextfile`	Stops processing the current file and immediately advances to the next file.
`print`	Prints quoted text, records, fields, and variables. (The default is to print the entire current record.)
`printf`	Prints formatted text, similar to its `C` counterpart, but the trailing newline must be specified.
`sprintf`	Returns as a string as formatted text using the same format as `printf`.

Section 4. Running GAWK programs

GAWK takes two input files -- the command file containing the AWK program itself and the data file(s) to work on. You need some sample data, so use a text editor to place the entire contents (three lines) of Listing 4 into a file called *sample.*

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 4. Sample text

```
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
```

Methods of running GAWK programs

You can run GAWK programs in any of four ways: at the command line, as a filter, from a file, or as a script.

At the command line

You don't have to keep commands in a file: You can pass them as arguments. This is the common method of running GAWK *one-liners*.

The format is:

```
awk 'program' filespec
```

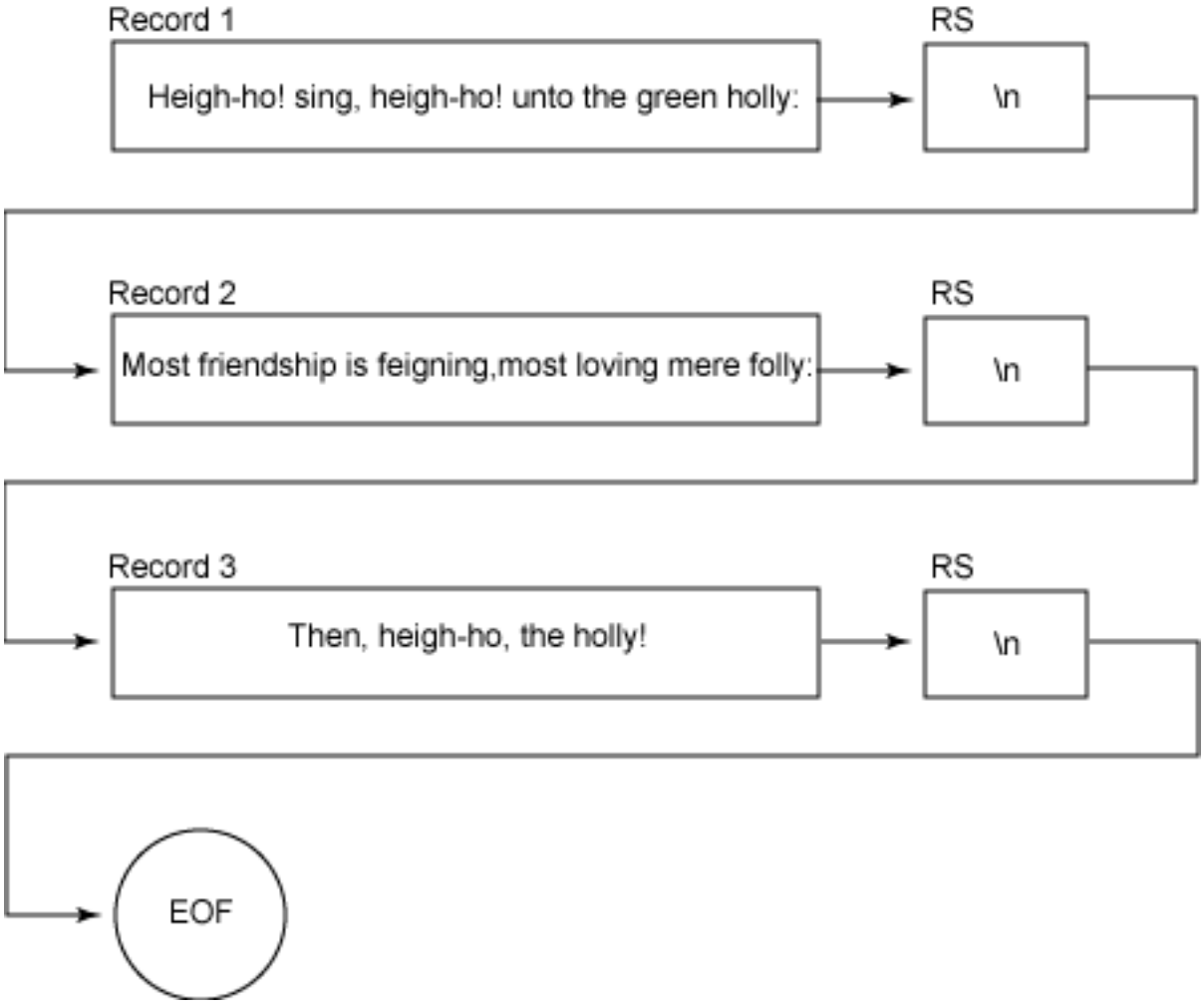
Try this one-liner to print every line of the sample file:

```
$ awk '{ print }' sample
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
$
```

This program reads in the file sample and prints each record -- in other words, it just outputs the entire file verbatim. To GAWK, the data in the file looked like Figure 3.

Figure 3. GAWK's interpretation of the sample file

© Copyright IBM Corporation 1994, 2008. All rights reserved.



Because GAWK takes multiple input files, this program concatenates multiple files if you specify them as arguments.

One-liners aren't limited to programs that fit on a single line. You could, for instance, quote all of Listing 3 on the command line, and the program runs the code in Listing 5.

Listing 5. A multiple-line one-liner

```
$ awk 'BEGIN { print "Beginning of file.";
print "-----------------" },
//
END { print "------------";
print "End of file."}' sample
Beginning of file
-----------------
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
------------
End of file.
$
```

Besides the practicality of typing programs of any length, running AWK programs at the command line has a caveat -- you must be careful with quoting! In AWK,

© Copyright IBM Corporation 1994, 2008. All rights reserved.

variables, such as $1, represent fields, as described in the next section; in many shells, these fields are special variables that translate to the arguments given at the command line.

As a filter

AWK is frequently used as a text filter in a pipeline. This is where GAWK reads its data from the standard input:

```
awk 'program'
```

Try testing it with your sample file:

```
$ cat sample | awk '{ print }'
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
$
```

From a file

In practice, only one-liners are ever run from the command line. Large GAWK programs are always in a file. Use the `-f` option to specify the program file.

So, assume that a file named *progfile* contains the following:

```
{ print }
```

You get the same output you would when running `awk` at the command line when you run this:

```
$ awk -f progfile sample
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
$
```

Notice that when you have the program in a file, there's no need for shell quoting.

As a script

Another way to run a GAWK program is to put it in a file and make it an executable script with the *shebang* command execution, if your shell supports it.

Put the program in a GAWK script file called *awkat,* make it executable, and try running it with the sample file, as shown in Listing 6.

Listing 6. Running the simple program as an executable

© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
$ cat > awkat
                    #!/usr/bin/awk -f
                    { print }
                    Ctrl-d
$ chmod u+x awkat
$ ./awkat sample
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
$
```

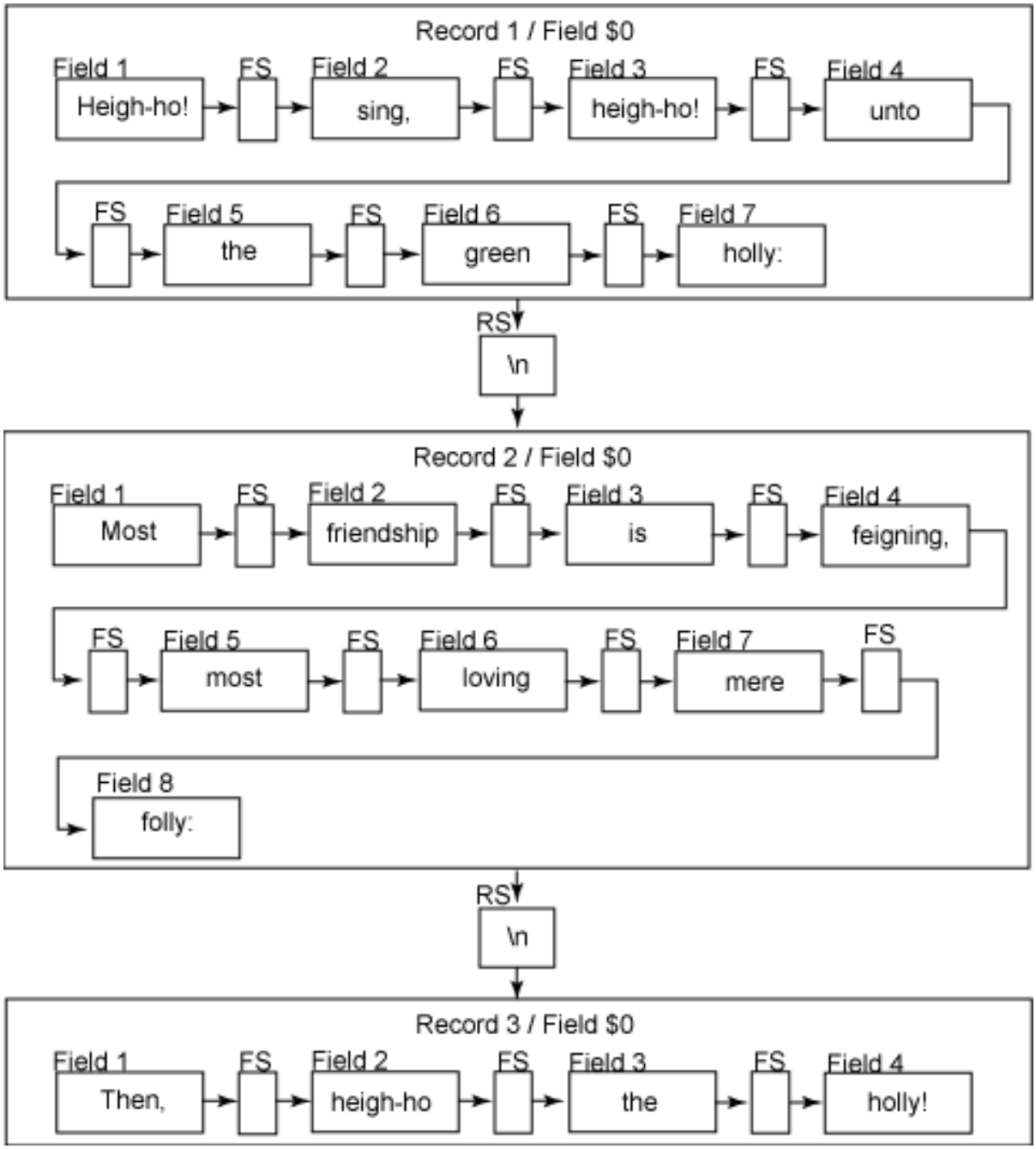
Section 5. Manipulating records and fields

When GAWK reads in a record, it stores all the fields of that record in variables. You can access each field by a $ followed by the field number -- so $1 references the first field, $2 references the second, and so on, all the way to the last field of the record.

Figure 4 shows the sample text delineated in the default records and fields.

Figure 4. Sample file broken down into AWK records and fields

© Copyright IBM Corporation 1994, 2008. All rights reserved.



As described in Elements of an input file, you can reference an entire record with `$0`, which includes all fields and field separators. This is the default value for many commands. So for example, `print`, as you've done before, is the same as `print $0` -- both commands print the entire current record.

Print specific fields

To output a certain field, give the name of that field as an argument to `print`. Try printing the first field in every record in the sample file:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
$ awk ' { print $1 } ' sample
Heigh-ho!
Most
Then,
$
```

You can give multiple fields in a `print` statement, and they can be in any order:

```
$ awk ' { print $7, $3, $1 } ' sample
holly: heigh-ho! Heigh-ho!
mere is Most
 the Then,
$
```

Notice that some lines don't have a seventh field; in such cases, nothing is printed.

When separated by a comma, the fields are output with spaces between them. You can concatenate them by omitting the comma. To print the seventh and eighth fields concatenated together, use:

```
$ awk ' { print $7 $8 } ' sample
holly:
merefolly:
$
```

You can combine quoted text with fields. Try this:

```
$ awk ' { print "Field 2: " $2 } ' sample
Field 2: sing,
Field 2: friendship
Field 2: heigh-ho,
$
```

You should already be getting an idea of the power of working on fields and records -- with AWK, tabular data is easy to parse, manipulate, and reformat using just a few simple commands. You can use shell redirection to direct the reformatted output to a new file, or pass it down a pipeline.

Working as a filter, this functionality becomes useful in conjunction with other commands. For example, this command modifies the default output of a date so that it prints in a *day month, year* format:

```
$ date|awk '{print $3 " " $2 ", " $6}'
29 Nov, 2006
$
```

Change the field separator

The fields in the examples so far have been separated by space characters. That's the default behavior -- any number of space or tab characters -- and you can change

© Copyright IBM Corporation 1994, 2008. All rights reserved.

it. The value of the field separator is contained in the FS variable. Like any variable in AWK, it can be redefined at any time in a program. To use a different field separator for the entire file, redefine it in a BEGIN statement.

Print the first field of the sample data with a field separator of an exclamation point (!):

```
$ awk ' BEGIN { FS = "!" } { print $1 } ' sample
Heigh-ho
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly
$
```

Note the differences when printing the second and third fields:

```
$ awk ' BEGIN { FS = "!" } { print $2 } ' sample
 sing, heigh-ho


$ awk ' BEGIN { FS = "!" } { print $3 } ' sample
 unto the green holly:


$
```

Try comparing the fields in the output you get with the fields listed in Figure 4.

But the field separator doesn't have to be a single character. Try using a phrase:

```
$ awk ' BEGIN { FS = "Heigh-ho" } { print $2 } ' sample
! sing, heigh-ho! unto the green holly:


$
```

In GAWK, the field separator can be any regular expression. To make each input character its own field, give FS a value of null.

Capitalization counts. The example above only matches one separator in the entire file, but the following example matches the same phrase regardless of case:

```
$ awk ' BEGIN { FS = "[Hh]eigh-ho" } { print $2 } ' sample
! sing,

, the holly!
$
```

You can also change the field separator from the command line by specifying it as a quoted argument to the -F option:

```
$ awk -F "," ' { print $2 } ' sample
 heigh-ho! unto the green holly:
 most loving mere folly:
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
 heigh-ho
$
```

This functionality makes it easy to create one-liners that can parse files, such as /etc/passwd, where fields are delimited by a colon (:) character. You can easily pull out a list of full user names, for example:

```
$ awk -F ":" ' { print $5 } ' /etc/passwd
```

Change the record separator

As with the field separator, you can change the record separator from its default -- a newline character -- to anything you'd like. Its current value is kept in the `RS` variable.

Change the record separator to a comma, and try it on the sample file:

```
$ awk ' BEGIN { RS = "," } //' sample
Heigh-ho! sing
 heigh-ho! unto the green holly:
Most friendship is feigning
 most loving mere folly:
Then
 heigh-ho
 the holly!
$
```

Change the output

AWK output is handled just like AWK input data and it is divided into fields and records; the output stream has its own separators, which are initially set to the same defaults as the input separators -- spaces and newlines. The *output field separator*, used in `print` statements in which fields are separated by commas, is set to a single space, and you can change it by redefining the `OFS` variable. The *output record separator* is set to a newline character, and you can change it by redefining the `ORS` variable.

To strip all newlines from a file and place all the file's text on a single line -- which is useful for certain kinds of textual analysis and filtering -- just change the output record separator to the null character.

Try it on the sample file:

```
$ awk 'BEGIN {ORS=""} //' sample
Heigh-ho! sing, heigh-ho! unto the green holly:Most friendship\
is feigning, most loving mere folly:Then, heigh-ho, the holly!$
```

Every newline is stripped out, including the last. The returning shell prompt is on the same line as the output data. To add a final newline, put it in an `END` rule:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
$  awk 'BEGIN {ORS=""} // { print } END {print "\n"}' sample
Heigh-ho! sing, heigh-ho! unto the green holly:Most friendship\
is feigning, most loving mere folly:Then, heigh-ho, the holly!
$
```

More GAWK variables

The NF variable contains the number of fields in the current record. Using NF references its numeric value, while using $NF references the contents of the actual field itself. So if a record has 100 fields, `print NF` outputs the integer 100, while `print $100` outputs the same thing as `print $NF` -- the contents of the last field in the record.

The NR variable, in turn, contains the number of the current record. When the first record is being read, its value is 1; when the second record is being read, it increments to 2, and so on. Use it in an END pattern to output the number of lines in the input:

```
$ awk 'END { print "Input contains " NR " lines." }' sample
Input contains 3 lines.
$
```

Note: If the `print` statement above had been placed in a BEGIN pattern, the program would have reported that its input contained 0 lines, because the value for NR at the time of execution would be 0, as no records of input would have been read yet.

Use $NR to print the field relative to the current record number:

```
$ awk '{ print NR, $NR }' sample
1 Heigh-ho!
2 friendship
3 the
$
```

Take a look again at Figure 4, and see the values for Field 1 in the first record, Field 2 in the second record, and Field 3 in the last record. Compare this with your program output.

Try listing the number of fields for each record and the value of the last field:

```
$ awk ' { print "Record " NR " has " NF " fields and ends with " $NF}' sample
Record 1 has 7 fields and ends with holly:
Record 2 has 8 fields and ends with folly:
Record 3 has 4 fields and ends with holly!
```

There are a handful of special GAWK variables that are frequently used. Table 2 lists them and describes their meaning.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Table 2. Common GAWK variables

Variable	Description
NF	This variable contains the number of fields per record.
NR	This variable contains the number of the current record.
FS	This variable is the field separator.
RS	This variable is the record separator.
OFS	This variable is the output field separator.
ORS	This variable is the output record separator.
FILENAME	This variable contains the name of input file being read.
IGNORECASE	When IGNORECASE is set to a non-null value, GAWK ignores case in pattern matching.

Section 6. GAWK pattern matching

In GAWK, the pattern-matching facility is similar to that of the `egrep` command. To output only records that match a pattern, give that pattern in the rule, enclosed in slash characters.

Match patterns in input records

To match a single pattern, give it in a rule:

```
$ awk '/green/ { print }' sample
Heigh-ho! sing, heigh-ho! unto the green holly:
$
```

You're not limited to strings -- the pattern can be any extended regular expression. Try matching records in the sample input that contain two exclamation points with any amount of text between them:

```
$ awk '/!.*!/' sample
Heigh-ho! sing, heigh-ho! unto the green holly:
$
```

Unlike `grep`, you can also specify hex and decimal values of ASCII characters in your search. To give hex values, use the format `\x`*NN* , where *NN* is the hex value of the character in question. To give decimal values, use the format `\`*xxx* , where *xxx* is the decimal value of the character.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

This functionality is good for matching extended or non-printing characters, such as the `BELL` character (a literal Ctrl-G, which you can match with `'/\007/'`) as the pattern. However, AWK does not work on binary files.

Match patterns in fields

To match a pattern in a specific field (as opposed to in the entire record), give the field and use the ~ operator, which means "contains"; and then give the pattern:

```
$field ~ /pattern/
```

The `!~` operator works in the opposite manner, and you can use it to match fields that *do not* contain a given pattern:

```
$field !~ /pattern/
```

Try printing only records in the sample data whose third field contains an `s` character:

```
$ awk '$3 ~ /s/' sample
Most friendship is feigning, most loving mere folly:
$
```

To print the full names of all users whose login shell is not `bash`, use:

```
$ awk 'BEGIN {FS=":"} $7 !~ /bash/ {print $5}' /etc/passwd
```

Use boolean operators

You can combine patterns with the boolean operators, as shown in Table 3, and you can combine boolean operators in a single rule.

Table 3. GAWK boolean operators

Operator	Description
&&	Logical AND operator
||	Logical OR operator
!	NOT operator

Try printing the fourth field of only those records from the sample data that match the `regexps` `in`, `the`, and `to`:

```
$ awk '/in/ && /the/ && /to/ {print $4}' sample
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
unto
$
```

Now, print records from the sample data that do not contain `holly` or that *do* contain `most`, while changing the output record separator to a hyphen:

```
$ awk 'BEGIN { OFS="-" } ! /holly/ || /most/ { print $1, $2 }' sample
Most-friendship
$
```

Use range patterns

Use the comma between two patterns to specify a range -- meaning all the text between both patterns and the patterns themselves. Unlike other searches, range patterns match text across records. It outputs the full records that contain part of the match. For instance, searching for a range of `Heigh` to `folly` outputs the text of the first two records:

```
$ awk '/Heigh/,/folly/' sample
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
$
```

Output whole paragraphs for matches

If you make the field separator the newline character and you make the record separator a null string, AWK treats entire paragraphs as single records. This effectively makes it a "paragraph `grep`," outputting whole paragraphs containing matches to your search. This functionality can be useful on files containing large texts.

Search for *green* in the sample file after redefining the separators. Without redefining them, this search outputs only the first record (which is the first line). But because the entire sample file is a single paragraph, the match outputs the whole file itself:

```
$ awk 'BEGIN { FS="\n"; RS="" } /green/' sample
Heigh-ho! sing, heigh-ho! unto the green holly:
Most friendship is feigning, most loving mere folly:
Then, heigh-ho, the holly!
$
```

Section 7. Wrap-up

In this tutorial, you've begun to use GAWK, the GNU Project's open source

© Copyright IBM Corporation 1994, 2008. All rights reserved.

implementation of the AWK programming language. You've learned the rudiments of the AWK programming language -- including how to run and structure programs and basic operations, such as field and record manipulation and pattern matching. There's a lot more to learn -- AWK is truly one of the great UNIX *power tools* -- but even at your first exposure, you've been able to feel the power of the AWK language as a tool for text manipulation.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Resources

Learn

- "Effective management of system logs" (developerWorks, March 2006): This article describes a simple application of AWK and Extensible Markup Language (XML) to show UNIX system data in a reader-friendly way.
- GAWK: Effective AWK Programming: Read the standard user manual for GAWK.
- AIX and UNIX: Visit the developerWorks AIX and UNIX zone to expand your UNIX skills.
- New to AIX and UNIX: Visit the New to AIX and UNIX page to learn more about AIX and UNIX.
- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.
- AIX 5L Wiki: A collaborative environment for technical information related to AIX.
- Podcasts: Tune in and catch up with IBM technical experts.

Get products and technologies

- GAWK source code: Download a free copy of the latest from the GNU Project FTP site.
- IBM AIX Toolbox for Linux® Applications: A collection of open source and GNU software built for AIX 5L for IBM Systems System p™ systems and IBM RS/6000® that includes GAWK.
- NAWK: Like GAWK, the original AT&T, distributed by Brian Kernighan, is also open source.
- IBM trial software: Build your next development project with software for download directly from developerWorks.

Discuss

- Participate in the AIX and UNIX forums:
  - AIX 5L -- technical
  - AIX for Developers Forum
  - Cluster Systems Management
  - IBM Support Assistant
  - Performance Tools -- technical
  - Virtualization -- technical
  - More AIX and UNIX forums

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- Participate in the developerWorks blogs and get involved in the developerWorks community.

About the author

Michael Stutz
Michael Stutz is author of *The Linux Cookbook* , which he also designed and typeset using only open source software. His research interests include digital publishing and the future of the book. He has used various UNIX operating systems for 20 years. You can reach him at stutz@dsl.org.

Trademarks

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Other company, product, or service names might be trademarks or service marks of others.

© Copyright IBM Corporation 1994, 2008. All rights reserved.