NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [**−f** *program-file*] [**−F***c*] [*program*] [*variable=value . . .*] [*filename. . .*]

DESCRIPTION

awk scans each of its input *filename*s for lines that match any of a set of patterns specified in *program*. The input *filename*s are read in order; the standard input is read if there are no *filename*s. The filename '**−**' means the standard input.

The set of patterns may either appear literally on the command line as *program*, or, by using the '**−f** *program-file*' option, the set of patterns may be in a *program-file*; a *program-file* of '**−**' means the standard input. If the*pr ogram* is specified on the command line, it should be enclosed in single quotes (') to protect it from the shell.

awk variables may be set on the command line using arguments of the form *variable=value*. This sets the **awk** variable *variable* to *value* before the first record of the next *filename* argument is read.

With each pattern in *program* there can be an associated action that will be performed when a line of a *filename* matches the pattern. See the discussion below for the format of input lines and the **awk** language. Each line in each input *filename* is matched ag ainst the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

OPTIONS

−f *program-file*
: Use the contents of *program-file* as the source for the *program*.

−F*c*
: Use the character *c* as the field separator (FS) character. See the discussion of FS below.

USAGE

Input Lines

An input line is made up of fields separated by white space. The field separator can be changed by using FS — see**Special V ariable Names** below. Fields are denoted**$1**, **$2**, and so forth. **$0** refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form

pattern **{** *action* **}**

A missing *action* means copy the line to the output; a missing *pattern* al ways matches.

Action Statements

An *action* is a sequence of *statement*s. A*statement* can be one of the follo wing:

if (*conditional* **)** *statement* [**else** *statement*]
while (*conditional* **)** *statement*
for (*expression* **;** *conditional* **;** *expression* **)** *statement*
break
continue
{ [*statement*] . . .**}**
variable=expression
print [*expression-list*] [> *expression*]
Sprintf *format* [**,** *expression-list*] [> *expression*]
next skip remaining patterns on this input line
exit skip the rest of the input

Format of the awk Language

*statement*s are terminated by semicolons, NEWLINE characters or right braces. An empty *expression-list* stands for the whole line.

expressions take on string or numeric values as appropriate, and are built using the operators +, −, ∗, /, **%**, and concatenation (indicated by a blank). The C operators **++ , − − , += , − = ,** ∗**= , /= ,** and **%=** are also available in expressions.

variable may be scalars, array elements (denoted *x* **[** *i* **]**) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric, providing a form of associative memory. String constants are quoted"... ".

The **print** statement prints its arguments on the standard output (or on a file if **>** *filename* is present), separated by the current output field separator, and terminated by the output record separator. The**printf** statement formats its expression list according to the format template *format* (see **printf**(3V) for a description of the formatting control characters).

Built In Functions

The built-in function **length** returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions **exp**, **log**, **sqrt**, and **int**, where **int** truncates its argument to an integer. '**substr(** *s***,** *m***,** *n* **)**' returns the *n*-character substring of *s* that begins at position *m*. '**sprintf (***for* - *mat***,** *expression***,** *expression***,** ...**)**' formats the expressions according to the **printf** format given by *format*, and returns the resulting string.

Patterns

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in **egrep** (see **grep**(1V)), Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

where a *relop* is any of the six relational operators in C, and a *matchop* is either **~** (contains) or **!~** (does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special pattern **BEGIN** may be used to capture control before the first input line is read, in which case **BEGIN** must be the first pattern. The special pattern **END** may be used to capture control after the last input line is read, in which case **END** must be the last pattern.

Special Variable Names

A single character *c* may be used to separate the fields by starting the program with

BEGIN {FS = "*c***" }**

or by using the **−F***c* option.

Other variable names with special meanings include **NF**, the number of fields in the current record; **NR**, the ordinal number of the current record; **FILENAME**, the name of the current input file; **OFS**, the output field separator (default blank); **ORS**, the output record separator (default NEWLINE); and **OFMT**, the output format for numbers (default **%.6g**).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END      { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

{ for (i = NF; i > 0; − −i) print $i }

Print all lines between start/stop pairs:

/start/, /stop/

Print all lines whose first field is different from previous one:

$1 != prev { print; prev = $1 }

SEE ALSO

grep(1V), **lex**(1), **sed**(1V), **printf**(3V)

A. V. Aho, B. W. Kerninghan, P. J. Weinberger, *The AWK Programming Language* Addison-Wesley, 1988.

NOTES

The **awk** command is not changed to support 8-bit symbol names, as this would produce **awk** source code that is not portable between systems.

BUGS

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

There is no escape sequence that prints a double-quote. A workaround is to use the **sprintf** (see **printf**(3V)) function to store the character into a variable by its ASCII sequence.

dq = sprintf("%c", 34)

Syntax errors result in the cryptic message ‘**awk: bailing out near line 1**’.