## NAME

**awk** - pattern-directed scanning and processing language

## SYNOPSIS

**awk** [**-safe**] [**-version**] [**-d**[*n*]] [**-F** *fs*] [**-v** *var=value*] [*prog* | **-f** *progfile*] *file ...*

## DESCRIPTION

**awk** scans each input *file* for lines that match any of a set of patterns specified literally in *prog* or in one or more files specified as **-f** *progfile*. With each pattern there can be an associated action that will be performed when a line of a *file* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. The file name '-' means the standard input. Any *file* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename.

The options are as follows:

**-d**[*n*] Debug mode. Set debug level to *n*, or 1 if *n* is not specified. A value greater than 1 causes **awk** to dump core on fatal errors.

**-F** *fs* Define the input field separator to be the regular expression *fs*.

**-f** *progfile*
Read program code from the specified file *progfile* instead of from the command line.

**-safe** Disable file output (**print** >, **print** >>), process creation (*cmd* | **getline**, **print** |, **system**) and access to the environment (*ENVIRON*; see the section on variables below). This is a first (and not very reliable) approximation to a "safe" version of **awk**.

**-version**
Print the version number of **awk** to standard output and exit.

**-v** *var=value*
Assign *value* to variable *var* before *prog* is executed; any number of **-v** options may be present.

The input is normally made up of input lines (records) separated by newlines, or by the value of *RS*. If *RS* is null, then any number of blank lines are used as the record separator, and newlines are used as field separators (in addition to the value of *FS*). This is convenient when working with multi-line records.

An input line is normally made up of fields separated by whitespace, or by the extended regular

expression *FS* as described below. The fields are denoted *$1*, *$2*, ..., while *$0* refers to the entire line. If *FS* is null, the input line is split into one field per character. While both gawk and mawk have the same behavior, it is unspecified in the IEEE Std 1003.1-2008 ("POSIX.1") standard. If *FS* is a single space, then leading and trailing blank and newline characters are skipped. Fields are delimited by one or more blank or newline characters. A blank character is a space or a tab. If *FS* is a single character, other than space, fields are delimited by each single occurrence of that character. The *FS* variable defaults to a single space.

Normally, any number of blanks separate fields. In order to set the field separator to a single blank, use the **-F** option with a value of ‘[ ]’. If a field separator of ‘t’ is specified, **awk** treats it as if ‘\t’ had been specified and uses <TAB> as the field separator. In order to use a literal ‘t’ as the field separator, use the **-F** option with a value of ‘[t]’.

A pattern-action statement has the form

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

A missing **{** *action* **}** means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

Newlines are permitted after a terminating statement or following a comma (‘,’), an open brace (‘{’), a logical AND (‘&&’), a logical OR (‘||’), after the ‘do’ or ‘else’ keywords, or after the closing parenthesis of an ‘if’, ‘for’, or ‘while’ statement. Additionally, a backslash (‘\’) can be used to escape a newline between tokens.

An action is a sequence of statements. A statement can be one of the following:

**if** (*expression*) *statement* [**else** *statement*]
**while** (*expression*) *statement*
**for** (*expression*; *expression*; *expression*) *statement*
**for** (*var* **in** *array*) *statement*
**do** *statement* **while** (*expression*)
**break**
**continue**
**{** [*statement* ...] **}**
*expression* # commonly *var* = *expression*
**print** [*expression-list*] [>*expression*]
**printf** *format* [*...*, *expression-list*] [>*expression*]
**return** [*expression*]
**next** # skip remaining patterns on this input line

**nextfile** # skip rest of this file, open next, start at top
**delete** *array***[***expression***]** # delete an array element
**delete** *array* # delete all elements of array
**exit** [*expression*] # exit immediately; status is *expression*

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for *$0*. String constants are quoted "", with the usual C escapes recognized within (see printf(1) for a complete list of these). Expressions take on string or numeric values as appropriate, and are built using the operators **+ - * / % ^** (exponentiation), and concatenation (indicated by whitespace). The operators **! ++ -- += -= *= /= %= ^= > >= < <= == != ?:** are also available in expressions. Variables 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; this allows for a form of associative memory. Multiple subscripts such as [i,j,k] are permitted; the constituents are concatenated, separated by the value of *SUBSEP* (see the section on variables below).

The **print** statement prints its arguments on the standard output (or on a file if >*file* or >>*file* is present or on a pipe if | *cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The **printf** statement formats its expression list according to the format (see printf(1)).

Patterns are arbitrary Boolean combinations (with **! || &&**) of regular expressions and relational expressions. **awk** supports extended regular expressions (EREs). See re_format(7) for more information on regular expressions. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators **~** and **!~**. /*re*/ is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

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

A relational expression is one of the following:

*expression matchop regular-expression*
*expression relop expression*
*expression* **in** *array-name*
**(***expr, expr, ...***) in** *array-name*

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

of these.

The special patterns **BEGIN** and **END** may be used to capture control before the first input line is read and after the last. **BEGIN** and **END** do not combine with other patterns.

Variable names with special meanings:

| | |
|---|---|
| *ARGC* | Argument count, assignable. |
| *ARGV* | Argument array, assignable; non-null members are taken as filenames. |
| *CONVFMT* | Conversion format when converting numbers (default "%.6g"). |
| *ENVIRON* | Array of environment variables; subscripts are names. |
| *FILENAME* | The name of the current input file. |
| *FNR* | Ordinal number of the current record in the current file. |
| *FS* | Regular expression used to separate fields; also settable by option **-F** *fs*. |
| *NF* | Number of fields in the current record. *$NF* can be used to obtain the value of the last field in the current record. |
| *NR* | Ordinal number of the current record. |
| *OFMT* | Output format for numbers (default "%.6g"). |
| *OFS* | Output field separator (default blank). |
| *ORS* | Output record separator (default newline). |
| *RLENGTH* | The length of the string matched by the **match**() function. |
| *RS* | Input record separator (default newline). |
| *RSTART* | The starting position of the string matched by the **match**() function. |
| *SUBSEP* | Separates multiple subscripts (default 034). |

## FUNCTIONS

The awk language has a variety of built-in functions: arithmetic, string, input/output, general, and bit-operation.

Functions may be defined (at the position of a pattern-action statement) thusly:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar, and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global. Thus local variables may be created by providing excess parameters in the function definition.

### Arithmetic Functions

**atan2**(*y*, *x*) Return the arctangent of *y*/*x* in radians.

**cos**(*x*) Return the cosine of *x*, where *x* is in radians.

**exp**(*x*) Return the exponential of *x*.

**int**(*x*) Return *x* truncated to an integer value.

**log**(*x*) Return the natural logarithm of *x*.

**rand**() Return a random number, *n*, such that 0<=*n*<1.

**sin**(*x*) Return the sine of *x*, where *x* is in radians.

**sqrt**(*x*) Return the square root of *x*.

**srand**(*expr*)
Sets seed for **rand**() to *expr* and returns the previous seed. If *expr* is omitted, the time of day is used instead.

**String Functions**

**gsub**(*r*, *t*, *s*) The same as **sub**() except that all occurrences of the regular expression are replaced. **gsub**() returns the number of replacements.

**index**(*s*, *t*) The position in *s* where the string *t* occurs, or 0 if it does not.

**length**(*s*) The length of *s* taken as a string, or of *$0* if no argument is given.

**match**(*s*, *r*) The position in *s* where the regular expression *r* occurs, or 0 if it does not. The variable *RSTART* is set to the starting position of the matched string (which is the same as the returned value) or zero if no match is found. The variable *RLENGTH* is set to the length of the matched string, or -1 if no match is found.

**split**(*s*, *a*, *fs*) Splits the string *s* into array elements *a[1], a[2], ..., a[n]* and returns *n*. The separation is done with the regular expression *fs* or with the field separator *FS* if *fs* is not given. An empty string as field separator splits the string into one array element per character.

**sprintf**(*fmt*, *expr*, ...)
The string resulting from formatting *expr*, ... according to the printf(1) format *fmt*.

**sub**(*r*, *t*, *s*) Substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, *$0* is used. An ampersand ('&') in *t* is replaced in string *s* with regular expression *r*.

A literal ampersand can be specified by preceding it with two backslashes (‘\\’). A literal backslash can be specified by preceding it with another backslash (‘\\’). **sub**() returns the number of replacements.

**substr**(*s*, *m*, *n*)
Return at most the *n*-character substring of *s* that begins at position *m* counted from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string, the length of the substring is limited by the length of *s*.

**tolower**(*str*) Returns a copy of *str* with all upper-case characters translated to their corresponding lower-case equivalents.

**toupper**(*str*) Returns a copy of *str* with all lower-case characters translated to their corresponding upper-case equivalents.

**Input/Output and General Functions**

**close**(*expr*) Closes the file or pipe *expr*. *expr* should match the string that was used to open the file or pipe.

*cmd* | **getline** [*var*]
Read a record of input from a stream piped from the output of *cmd*. If *var* is omitted, the variables *$0* and *NF* are set. Otherwise *var* is set. If the stream is not open, it is opened. As long as the stream remains open, subsequent calls will read subsequent records from the stream. The stream remains open until explicitly closed with a call to **close**(). **getline** returns 1 for a successful input, 0 for end of file, and -1 for an error.

**fflush**(*[expr]*) Flushes any buffered output for the file or pipe *expr*, or all open files or pipes if *expr* is omitted. *expr* should match the string that was used to open the file or pipe.

**getline** Sets *$0* to the next input record from the current input file. This form of **getline** sets the variables *NF*, *NR*, and *FNR*. **getline** returns 1 for a successful input, 0 for end of file, and -1 for an error.

**getline** *var* Sets *$0* to variable *var*. This form of **getline** sets the variables *NR* and *FNR*. **getline** returns 1 for a successful input, 0 for end of file, and -1 for an error.

**getline** [*var*] <*file* Sets *$0* to the next record from *file*. If *var* is omitted, the variables *$0* and *NF* are set. Otherwise *var* is set. If *file* is not open, it is opened. As long as the stream remains open, subsequent calls will read subsequent records from *file*. *file* remains

open until explicitly closed with a call to **close**().

**system**(*cmd*) Executes *cmd* and returns its exit status.

**Bit-Operation Functions**

**compl**(*x*) Returns the bitwise complement of integer argument x.

**and**(*v1*, *v2*, ...)
Performs a bitwise AND on all arguments provided, as integers. There must be at least two values.

**or**(*v1*, *v2*, ...)
Performs a bitwise OR on all arguments provided, as integers. There must be at least two values.

**xor**(*v1*, *v2*, ...)
Performs a bitwise Exclusive-OR on all arguments provided, as integers. There must be at least two values.

**lshift**(*x*, *n*)
Returns integer argument x shifted by n bits to the left.

**rshift**(*x*, *n*)
Returns integer argument x shifted by n bits to the right.

## EXIT STATUS

The **awk** utility exits 0 on success, and >0 if an error occurs.

But note that the **exit** expression can modify the exit status.

## EXAMPLES

Print lines longer than 72 characters:

```
length($0) > 72
```

Print first two fields in opposite order:

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

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
    { print $2, $1 }
```

Add up first column, print sum and average:

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

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Simulate echo(1):

```
BEGIN { # Simulate echo(1)
      for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
      printf "\n"
      exit }
```

Print an error message to standard error:

```
{ print "error!" > "/dev/stderr" }
```

## SEE ALSO

cut(1), lex(1), printf(1), sed(1), re_format(7)

A. V. Aho, B. W. Kernighan, and P. J. Weinberger, *The AWK Programming Language*, *Addison-Wesley*, 1988, ISBN 0-201-07981-X.

## STANDARDS

The **awk** utility is compliant with the IEEE Std 1003.1-2008 ("POSIX.1") specification, except **awk** does not support {n,m} pattern matching.

The flags **-d**, **-safe**, and **-version** as well as the commands **fflush**, **compl**, **and**, **or**, **xor**, **lshift**, **rshift**, are extensions to that specification.

## HISTORY

An **awk** utility appeared in Version 7 AT&T UNIX.

## BUGS

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 "" to it.

The scope rules for variables in functions are a botch; the syntax is worse.

## DEPRECATED BEHAVIOR

One True Awk has accpeted **-F** *t* to mean the same as **-F** <*TAB*> to make it easier to specify tabs as the separator character. Upstream One True Awk has deprecated this wart in the name of better compatibility with other awk implementations like gawk and mawk.

Historically, **awk** did not accept "0x" as a hex string. However, since One True Awk used strtod to convert strings to floats, and since "0x12" is a valid hexadecimal representation of a floating point number, On FreeBSD, **awk** has accepted this notation as an extension since One True Awk was imported in FreeBSD 5.0. Upstream One True Awk has restored the historical behavior for better compatibility between the different awk implementations. Both gawk and mawk already behave similarly. Starting with FreeBSD 14.0 **awk** will no longer accept this extension.

The FreeBSD **awk** sets the locale for many years to match the environment it was running in. This lead to pattern ranges, like "[A-Z]" sometimes matching lower case characters in some locales. This misbehavior was never in upstream One True Awk and has been removed as a bug in FreeBSD 12.3, FreeBSD 13.1, and FreeBSD 14.0.