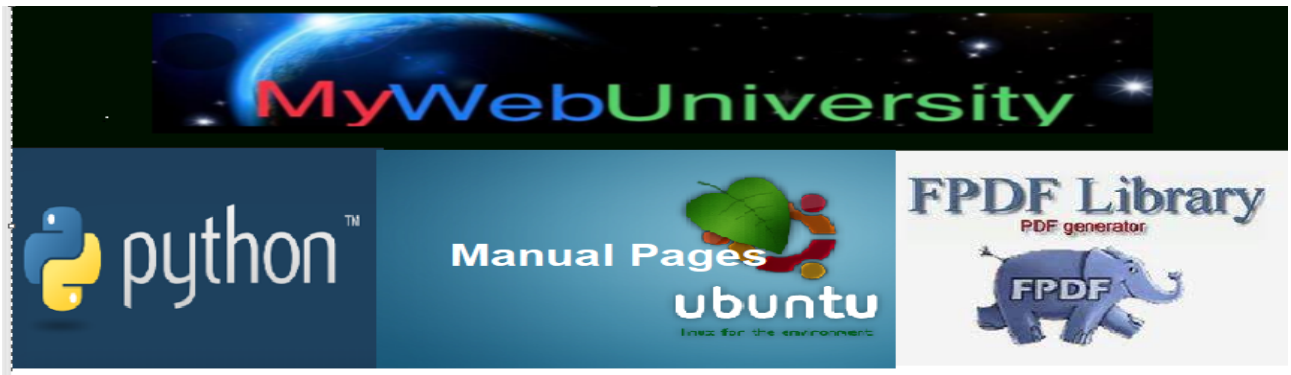

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'awk.1'

$ man awk.1

GAWK(1) Utility Commands GAWK(1)

NAME

gawk - pattern scanning and processing language

SYNOPSIS

gawk [POSIX or GNU style options] -f program-file [--] file ...

gawk [POSIX or GNU style options] [--] program-text file ...

README FIRST

This manual page is provided as a courtesy. Please note that the One Source Of Truth for gawk is the Texinfo manual, available online in several formats at https://www.gnu.org/soft?ware/gawk/manual. It may also be installed in the Info subsystem on your system, and avail?able therefore via the info(1) command.

In the case of any contradiction between the Texinfo manual and this man page, the manual should be considered to be authoritative.

DESCRIPTION

Gawk is the GNU Project's implementation of the AWK programming language. It conforms to the definition of the language in the POSIX 1003.1 standard. This version in turn is based on the description in The AWK Programming Language, by Aho, Kernighan, and Weinberger. Gawk provides the additional features found in the current version of Brian Kernighan's awk and numerous GNU-specific extensions.

The command line consists of options to gawk itself, the AWK program text (if not supplied via the -f or --include options), and values to be made available in the ARGC and ARGV pre-defined AWK variables.

PREFACE

This manual page is intentionally as terse as possible. Full details are provided in GAWK: Effective AWK Programming, and you should look there for the full story on any specific fea? ture. Where possible, links to the online version of the manual are provided.

OPTION FORMAT

Gawk options may be either traditional POSIX-style one letter options, or GNU-style long op? tions. POSIX options start with a single ?-?, while long options start with ?--?. Long op? tions are provided for both GNU-specific features and for POSIX-mandated features.

Gawk-specific options are typically used in long-option form. Arguments to long options are either joined with the option by an = sign, with no intervening spaces, or they may be pro? vided in the next command line argument. Long options may be abbreviated, as long as the abbreviation remains unique.

Additionally, every long option has a corresponding short option, so that the option's func? tionality may be used from within #! executable scripts.

OPTIONS

Gawk accepts the following options. Standard options are listed first, followed by options for gawk extensions, listed alphabetically by short option.

-f program-file, --file program-file

Read the AWK program source from the file program-file, instead of from the first command line argument. Multiple -f options may be used. Files read with -f are treated as if they begin with an implicit @namespace "awk" statement.

-F fs, --field-separator fs

Use fs for the input field separator (the value of the FS predefined variable).

-v var=val, --assign var=val

Assign the value val to the variable var, before execution of the program begins. Such variable values are available to the BEGIN rule of an AWK program.

-b, --characters-as-bytes

Treat all input data as single-byte characters. The --posix option overrides this one.

-c, --traditional

Run in compatibility mode. In compatibility mode, gawk behaves identically to Brian Kernighan's awk; none of the GNU-specific extensions are recognized.

-C, --copyright

Print the short version of the GNU copyright information message on the standard out?

put and exit successfully.

-d[file], --dump-variables[=file]

Print a sorted list of global variables, their types and final values to file. The default file is awkvars.out in the current directory.

-D[file], --debug[=file]

Enable debugging of AWK programs. By default, the debugger reads commands interactively from the keyboard (standard input). The optional file argument specifies a file with a list of commands for the debugger to execute non-interactively.

In this mode of execution, gawk loads the AWK source code and then prompts for debugging commands. Gawk can only debug AWK program source provided with the -f and --include options. The debugger is documented in GAWK: Effective AWK Programming; see https://www.gnu.org/software/gawk/manual/html_node/Debugger.html#Debugger.

-e program-text, --source program-text

Use program-text as AWK program source code. Each argument supplied via -e is treated as if it begins with an implicit @namespace "awk" statement.

-E file, --exec file

Similar to -f, however, this option is the last one processed. This should be used with #! scripts, particularly for CGI applications, to avoid passing in options or source code (!) on the command line from a URL. This option disables command-line variable assignments.

-g, --gen-pot

Scan and parse the AWK program, and generate a GNU .pot (Portable Object Template) format file on standard output with entries for all localizable strings in the program. The program itself is not executed.

-h, --help

Print a relatively short summary of the available options on the standard output.

Per the GNU Coding Standards, these options cause an immediate, successful exit.

-i include-file, --include include-file

Load an awk source library. This searches for the library using the AWKPATH environment variable. If the initial search fails, another attempt will be made after appending the .awk suffix. The file will be loaded only once (i.e., duplicates are eliminated), and the code does not constitute the main program source. Files read with --include are treated as if they begin with an implicit @namespace "awk" state?

ment.

-I, --trace

Print the internal byte code names as they are executed when running the program. The trace is printed to standard error. Each ?op code? is preceded by a + sign in the output.

-k, --csv

Enable CSV special processing. See Comma Separated Values, below, for more detail.

-l lib, --load lib

Load a gawk extension from the shared library lib. This searches for the library us? ing the AWKLIBPATH environment variable. If the initial search fails, another at? tempt will be made after appending the default shared library suffix for the plat? form. The library initialization routine is expected to be named dl_load().

-L [value], --lint[=value]

Provide warnings about constructs that are dubious or non-portable to other AWK im? plementations. See https://www.gnu.org/software/gawk/manual/html_node/Op? tions.html#Options for the list of possible values for value.

-M, --bignum

Force arbitrary precision arithmetic on numbers. This option has no effect if gawk is not compiled to use the GNU MPFR and GMP libraries. (In such a case, gawk issues a warning.)

NOTE: This feature is on parole. The primary gawk maintainer is no longer supporting it, although there is a member of the development team who is. If this situation changes, the feature will be removed from gawk.

-n, --non-decimal-data

Recognize octal and hexadecimal values in input data. Use this option with great caution!

-N, --use-lc-numeric

Force gawk to use the locale's decimal point character when parsing input data.

-o[file], --pretty-print[=file]

Output a pretty printed version of the program to file. The default file is awkprof.out in the current directory. This option implies --no-optimize.

-O, --optimize

Enable gawk's default optimizations upon the internal representation of the program.

This option is on by default.

-p[prof-file], --profile[=prof-file]

Start a profiling session, and send the profiling data to prof-file. The default is awkprof.out in the current directory. The profile contains execution counts of each statement in the program in the left margin and function call counts for each user-defined function. Gawk runs more slowly in this mode. This option implies --no-optimize.

-P, --posix

This turns on compatibility mode, and disables a number of common extensions.

-r, --re-interval

Enable the use of interval expressions in regular expression matching. Interval expressions are enabled by default, but this option remains for backwards compatibility.

-s, --no-optimize

Disable gawk's default optimizations upon the internal representation of the program.

-S, --sandbox

Run gawk in sandbox mode, disabling the system() function, input redirection with getline, output redirection with print and printf, and loading dynamic extensions. Command execution (through pipelines) is also disabled.

-t, --lint-old

Provide warnings about constructs that are not portable to the original version of UNIX awk.

-V, --version

Print version information for this particular copy of gawk on the standard output. This is useful when reporting bugs. Per the GNU Coding Standards, these options cause an immediate, successful exit.

-- Signal the end of options. This is useful to allow further arguments to the AWK program itself to start with a -.

In compatibility mode, any other options are flagged as invalid, but are otherwise ignored. In normal operation, as long as program text has been supplied, unknown options are passed on to the AWK program in the ARGV array for processing.

For POSIX compatibility, the -W option may be used, followed by the name of a long option.

An AWK program consists of a sequence of optional directives, pattern-action statements, and optional function definitions.

```
@include "filename"
@load "filename"
@namespace "name"
pattern   { action statements }
function name(parameter list) { statements }
```

Gawk first reads the program source from the program-file(s) if specified, from arguments to --source, or from the first non-option argument on the command line. The -f and --source options may be used multiple times on the command line. Gawk reads the program text as if all the program-files and command line source texts had been concatenated together.

In addition, lines beginning with @include may be used to include other source files into your program. This is equivalent to using the --include option.

Lines beginning with @load may be used to load extension functions into your program. This is equivalent to using the --load option.

The environment variable AWKPATH specifies a search path to use when finding source files named with the -f and --include options. If this variable does not exist, the default path is ".:/usr/local/share/awk". (The actual directory may vary, depending upon how gawk was built and installed.) If a file name given to the -f option contains a ?/? character, no path search is performed.

The environment variable AWKLIBPATH specifies a search path to use when finding source files named with the --load option. If this variable does not exist, the default path is "/usr/local/lib/gawk". (The actual directory may vary, depending upon how gawk was built and installed.)

Gawk executes AWK programs in the following order. First, all variable assignments speci? fied via the -v option are performed. Next, gawk compiles the program into an internal form. Then, gawk executes the code in the BEGIN rule(s) (if any), and then proceeds to read each file named in the ARGV array (up to ARGV[ARGC-1]). If there are no files named on the command line, gawk reads the standard input.

If a filename on the command line has the form var=val it is treated as a variable assign? ment. The variable var will be assigned the value val. (This happens after any BEGIN rule(s) have been run.)

If the value of a particular element of ARGV is empty (""), gawk skips over it.

For each input file, if a BEGINFILE rule exists, gawk executes the associated code before processing the contents of the file. Similarly, gawk executes the code associated with END? FILE rules after processing the file.

For each record in the input, gawk tests to see if it matches any pattern in the AWK pro? gram. For each pattern that the record matches, gawk executes the associated action. The patterns are tested in the order they occur in the program.

Finally, after all the input is exhausted, gawk executes the code in the END rule(s) (if any).

Command Line Directories

According to POSIX, files named on the awk command line must be text files. The behavior is ?undefined? if they are not. Most versions of awk treat a directory on the command line as a fatal error.

For gawk, a directory on the command line produces a warning, but is otherwise skipped. If either of the --posix or --traditional options is given, then gawk reverts to treating di? rectories on the command line as a fatal error.

VARIABLES, RECORDS AND FIELDS

AWK variables are dynamic; they come into existence when they are first used. Their values are either floating-point numbers or strings, or both, depending upon how they are used. Additionally, gawk allows variables to have regular-expression type. AWK also has one di? mensional arrays; arrays with multiple dimensions may be simulated. However, gawk provides true arrays of arrays. Several pre-defined variables are set as a program runs; these are described as needed and summarized below.

Records

Normally, records are separated by newline characters. You can control how records are sep? arated by assigning values to the built-in variable RS. See https://www.gnu.org/soft? ware/gawk/manual/html_node/Records.html for the details.

Fields

As each input record is read, gawk splits the record into fields, using the value of the FS variable as the field separator. Additionally, FIELDWIDTHS and FPAT may be used to control input field splitting. See the details, starting at https://www.gnu.org/software/gawk/man? ual/html_node/Fields.html.

Each field in the input record may be referenced by its position: $1, $2, and so on. $0 is the whole record, including leading and trailing whitespace.

The variable NF is set to the total number of fields in the input record.

References to non-existent fields (i.e., fields after $NF) produce the null string. How? ever, assigning to a non-existent field (e.g., $(NF+2) = 5) increases the value of NF, cre? ates any intervening fields with the null string as their values, and causes the value of $0 to be recomputed, with the fields being separated by the value of OFS. References to nega? tive numbered fields cause a fatal error. Decrementing NF causes the values of fields past the new value to be lost, and the value of $0 to be recomputed, with the fields being sepa? rated by the value of OFS.

Assigning a value to an existing field causes the whole record to be rebuilt when $0 is ref? erenced. Similarly, assigning a value to $0 causes the record to be resplit, creating new values for the fields.

Comma Separated Values

When invoked with either the -k or the --csv option, gawk does not use regular record deter? mination and field splitting as described above. Instead, records are terminated by un? quoted newlines, and fields are separated by commas. Double-quotes may be used to enclose fields containing commas, newlines, or doubled double-quotes. See https://www.gnu.org/soft? ware/gawk/manual/html_node/Comma-Separated-Fields.html for more details.

Built-in Variables

Gawk's built-in variables are listed below. This list is purposely terse. For details, see https://www.gnu.org/software/gawk/manual/html_node/Built_002din-Variables.

ARGC The number of command line arguments.

ARGIND The index in ARGV of the current file being processed.

ARGV Array of command line arguments. The array is indexed from 0 to ARGC - 1.

BINMODE On non-POSIX systems, specifies use of ?binary? mode for all file I/O. See https://www.gnu.org/software/gawk/manual/html_node/PC-Using.html for the de? tails.

CONVFMT The conversion format for numbers, "%.6g", by default.

ENVIRON An array containing the values of the current environment. The array is indexed by the environment variables, each element being the value of that variable.

ERRNO If a system error occurs either doing a redirection for getline, during a read for getline, or during a close(), then ERRNO is set to a string describing the error. The value is subject to translation in non-English locales.

FIELDWIDTHS A whitespace-separated list of field widths. When set, gawk parses the input

into fields of fixed width, instead of using the value of the FS variable as the field separator. Each field width may optionally be preceded by a colon-sepa? rated value specifying the number of characters to skip before the field starts.

FILENAME The name of the current input file. If no files are specified on the command line, the value of FILENAME is ?-?. However, FILENAME is undefined inside the BEGIN rule (unless set by getline).

FNR The input record number in the current input file.

FPAT A regular expression describing the contents of the fields in a record. When set, gawk parses the input into fields, where the fields match the regular ex? pression, instead of using the value of FS as the field separator.

FS The input field separator, a space by default. See https://www.gnu.org/soft? ware/gawk/manual/html_node/Field-Separators.html for the details.

FUNCTAB An array whose indices and corresponding values are the names of all the user-defined or extension functions in the program. NOTE: You may not use the delete statement with the FUNCTAB array.

IGNORECASE Controls the case-sensitivity of all regular expression and string operations. See https://www.gnu.org/software/gawk/manual/html_node/Case_002dsensitivity.html for details.

LINT Provides dynamic control of the --lint option from within an AWK program.

NF The number of fields in the current input record.

NR The total number of input records seen so far.

OFMT The output format for numbers, "%.6g", by default.

OFS The output field separator, a space by default.

ORS The output record separator, by default a newline.

PREC The working precision of arbitrary precision floating-point numbers, 53 by de? fault.

PROCINFO The elements of this array provide access to information about the running AWK program. See https://www.gnu.org/software/gawk/manual/html_node/Auto_002dset for the details.

ROUNDMODE The rounding mode to use for arbitrary precision arithmetic on numbers, by de? fault "N" (IEEE-754 roundTiesToEven mode). See https://www.gnu.org/soft? ware/gawk/manual/html_node/Setting-the-rounding-mode for the details.

RS The input record separator, by default a newline.

RT The record terminator. Gawk sets RT to the input text that matched the charac?
ter or regular expression specified by RS.

RSTART The index of the first character matched by match(); 0 if no match.

RLENGTH The length of the string matched by match(); -1 if no match.

SUBSEP The string used to separate multiple subscripts in array elements, by default
"\034".

SYMTAB An array whose indices are the names of all currently defined global variables
and arrays in the program. You may not use the delete statement with the SYMTAB
array, nor assign to elements with an index that is not a variable name.

TEXTDOMAIN The text domain of the AWK program; used to find the localized translations for
the program's strings.

Arrays

Arrays are subscripted with an expression between square brackets ([and]). If the expres?
sion is an expression list (expr, expr ...) then the array subscript is a string consisting
of the concatenation of the (string) value of each expression, separated by the value of the
SUBSEP variable. This facility is used to simulate multiply dimensioned arrays. For exam?
ple:

```
i = "A"; j = "B"; k = "C"
x[i, j, k] = "hello, world\n"
```

assigns the string "hello, world\n" to the element of the array x which is indexed by the
string "A\034B\034C". All arrays in AWK are associative, i.e., indexed by string values.
The special operator in may be used to test if an array has an index consisting of a partic?
ular value:

```
if (val in array)
    print array[val]
```

If the array has multiple subscripts, use (i, j) in array.
The in construct may also be used in a for loop to iterate over all the elements of an ar?
ray. However, the (i, j) in array construct only works in tests, not in for loops.
An element may be deleted from an array using the delete statement. The delete statement
may also be used to delete the entire contents of an array, just by specifying the array
name without a subscript.
gawk supports true multidimensional arrays. It does not require that such arrays be ?rec?
tangular? as in C or C++. See https://www.gnu.org/software/gawk/manual/html_node/Arrays for

details.

Namespaces

Gawk provides a simple namespace facility to help work around the fact that all variables in AWK are global.

A qualified name consists of a two simple identifiers joined by a double colon (::). The left-hand identifier represents the namespace and the right-hand identifier is the variable within it. All simple (non-qualified) names are considered to be in the ?current? name? space; the default namespace is awk. However, simple identifiers consisting solely of up? percase letters are forced into the awk namespace, even if the current namespace is differ? ent.

You change the current namespace with an @namespace "name" directive.

The standard predefined builtin function names may not be used as namespace names. The names of additional functions provided by gawk may be used as namespace names or as simple identifiers in other namespaces. For more details, see https://www.gnu.org/soft? ware/gawk/manual/html_node/Namespaces.html#Namespaces.

Variable Typing And Conversion

Variables and fields may be (floating point) numbers, or strings, or both. They may also be regular expressions. How the value of a variable is interpreted depends upon its context.

If used in a numeric expression, it will be treated as a number; if used as a string it will be treated as a string.

To force a variable to be treated as a number, add zero to it; to force it to be treated as a string, concatenate it with the null string.

Uninitialized variables have the numeric value zero and the string value "" (the null, or empty, string).

When a string must be converted to a number, the conversion is accomplished using strtod(3).

A number is converted to a string by using the value of CONVFMT as a format string for sprintf(3), with the numeric value of the variable as the argument. However, even though all numbers in AWK are floating-point, integral values are always converted as integers.

Gawk performs comparisons as follows: If two variables are numeric, they are compared numer? ically. If one value is numeric and the other has a string value that is a ?numeric string,? then comparisons are also done numerically. Otherwise, the numeric value is con? verted to a string and a string comparison is performed. Two strings are compared, of course, as strings.

Note that string constants, such as "57", are not numeric strings, they are string con? stants. The idea of ?numeric string? only applies to fields, getline input, FILENAME, ARGV elements, ENVIRON elements and the elements of an array created by split() or patsplit() that are numeric strings. The basic idea is that user input, and only user input, that looks numeric, should be treated that way.

Octal and Hexadecimal Constants

You may use C-style octal and hexadecimal constants in your AWK program source code. For example, the octal value 011 is equal to decimal 9, and the hexadecimal value 0x11 is equal to decimal 17.

String Constants

String constants in AWK are sequences of characters enclosed between double quotes (like "value"). Within strings, certain escape sequences are recognized, as in C. See https://www.gnu.org/software/gawk/manual/html_node/Escape-Sequences for the details.

Regexp Constants

A regular expression constant is a sequence of characters enclosed between forward slashes (like /value/).

The escape sequences described in the manual may also be used inside constant regular ex? pressions (e.g., /[\t\f\n\r\v]/ matches whitespace characters).

Gawk provides strongly typed regular expression constants. These are written with a leading @ symbol (like so: @/value/). Such constants may be assigned to scalars (variables, array elements) and passed to user-defined functions. Variables that have been so assigned have regular expression type.

PATTERNS AND ACTIONS

AWK is a line-oriented language. The pattern comes first, and then the action. Action statements are enclosed in { and }. Either the pattern may be missing, or the action may be missing, but, of course, not both. If the pattern is missing, the action executes for every single record of input. A missing action is equivalent to

```
{ print }
```

which prints the entire record.

Comments begin with the # character, and continue until the end of the line. Empty lines may be used to separate statements. Normally, a statement ends with a newline, however, this is not the case for lines ending in a comma, {, ?, :, &&, or ||. Lines ending in do or else also have their statements automatically continued on the following line. In other

cases, a line can be continued by ending it with a ?\?, in which case the newline is ig? nored. However, a ?\? after a # is not special.

Multiple statements may be put on one line by separating them with a ?;?. This applies to both the statements within the action part of a pattern-action pair (the usual case), and to the pattern-action statements themselves.

Patterns

AWK patterns may be one of the following:

BEGIN

END

BEGINFILE

ENDFILE

/regular expression/

relational expression

pattern && pattern

pattern || pattern

pattern ? pattern : pattern

(pattern)

! pattern

pattern1, pattern2

BEGIN and END are two special kinds of patterns which are not tested against the input. The action parts of all BEGIN patterns are merged as if all the statements had been written in a single BEGIN rule. They are executed before any of the input is read. Similarly, all the END rules are merged, and executed when all the input is exhausted (or when an exit state? ment is executed). BEGIN and END patterns cannot be combined with other patterns in pattern expressions. BEGIN and END patterns cannot have missing action parts.

BEGINFILE and ENDFILE are additional special patterns whose actions are executed before reading the first record of each command-line input file and after reading the last record of each file. Inside the BEGINFILE rule, the value of ERRNO is the empty string if the file was opened successfully. Otherwise, there is some problem with the file and the code should use nextfile to skip it. If that is not done, gawk produces its usual fatal error for files that cannot be opened.

For /regular expression/ patterns, the associated statement is executed for each input record that matches the regular expression. Regular expressions are essentially the same as

those in egrep(1). See https://www.gnu.org/software/gawk/manual/html_node/Regexp.html for the details on regular expressions.

A relational expression may use any of the operators defined below in the section on ac? tions. These generally test whether certain fields match certain regular expressions.

The &&, ||, and ! operators are logical AND, logical OR, and logical NOT, respectively, as in C. They do short-circuit evaluation, also as in C, and are used for combining more prim? itive pattern expressions. As in most languages, parentheses may be used to change the or? der of evaluation.

The ?: operator is like the same operator in C. If the first pattern is true then the pat? tern used for testing is the second pattern, otherwise it is the third. Only one of the second and third patterns is evaluated.

The pattern1, pattern2 form of an expression is called a range pattern. It matches all in? put records starting with a record that matches pattern1, and continuing until a record that matches pattern2, inclusive. It does not combine with any other sort of pattern expression.

Actions

Action statements are enclosed in braces, { and }. Action statements consist of the usual assignment, conditional, and looping statements found in most languages. The operators, control statements, and input/output statements available are patterned after those in C.

Operators

The operators in AWK, in order of decreasing precedence, are:

(...) Grouping

$ Field reference.

++ -- Increment and decrement, both prefix and postfix.

^ Exponentiation.

+ - ! Unary plus, unary minus, and logical negation.

* / % Multiplication, division, and modulus.

+ - Addition and subtraction.

space String concatenation.

| |& Piped I/O for getline, print, and printf.

< > <= >= == !=

The regular relational operators.

~ !~ Regular expression match, negated match.

in Array membership.

&& Logical AND.

|| Logical OR.

?: The C conditional expression. This has the form expr1 ? expr2 : expr3. If expr1 is true, the value of the expression is expr2, otherwise it is expr3. Only one of expr2 and expr3 is evaluated.

= += -= *= /= %= ^=

Assignment. Both absolute assignment (var = value) and operator-assignment (the other forms) are supported.

Control Statements

The control statements are as follows:

```
if (condition) statement [ else statement ]
while (condition) statement
do statement while (condition)
for (expr1; expr2; expr3) statement
for (var in array) statement
break
continue
delete array[index]
delete array
exit [ expression ]
{ statements }
switch (expression) {
case value|regex : statement
...
[ default: statement ]
}
```

I/O Statements

The input/output statements are as follows:

close(file [, how]) Close an open file, pipe or coprocess. The optional how should only be used when closing one end of a two-way pipe to a coprocess. It must be a string value, either "to" or "from".

getline Set $0 from the next input record; set NF, NR, FNR, RT.

getline <file Set $0 from the next record of file; set NF, RT.

getline var Set var from the next input record; set NR, FNR, RT.

getline var <file Set var from the next record of file; set RT.

command | getline [var]

Run command, piping the output either into $0 or var, as above, and RT.

command |& getline [var]

Run command as a coprocess piping the output either into $0 or var, as above, and RT. (The command can also be a socket. See the subsection Special File Names, below.)

fflush([file]) Flush any buffers associated with the open output file or pipe file. If file is missing or if it is the null string, then flush all open output files and pipes.

next Stop processing the current input record. Read the next input record and start processing over with the first pattern in the AWK program. Upon reaching the end of the input data, execute any END rule(s).

nextfile Stop processing the current input file. The next input record read comes from the next input file. Update FILENAME and ARGIND, reset FNR to 1, and start processing over with the first pattern in the AWK pro? gram. Upon reaching the end of the input data, execute any ENDFILE and END rule(s).

print Print the current record. The output record is terminated with the value of ORS.

print expr-list Print expressions. Each expression is separated by the value of OFS. The output record is terminated with the value of ORS.

print expr-list >file Print expressions on file. Each expression is separated by the value of OFS. The output record is terminated with the value of ORS.

printf fmt, expr-list Format and print.

printf fmt, expr-list >file

Format and print on file.

system(cmd-line) Execute the command cmd-line, and return the exit status. (This may not be available on non-POSIX systems.) See https://www.gnu.org/soft? ware/gawk/manual/html_node/I_002fO-Functions.html#I_002fO-Functions for the full details on the exit status.

Additional output redirections are allowed for print and printf.

print ... >> file

Append output to the file.

print ... | command

Write on a pipe.

print ... |& command

Send data to a coprocess or socket. (See also the subsection Special File Names, be?

low.)

The getline function returns 1 on success, zero on end of file, and -1 on an error. If the errno(3) value indicates that the I/O operation may be retried, and PROCINFO["input", "RETRY"] is set, then -2 is returned instead of -1, and further calls to getline may be at? tempted. Upon an error, ERRNO is set to a string describing the problem.

NOTE: Failure in opening a two-way socket results in a non-fatal error being returned to the calling function. If using a pipe, coprocess, or socket to getline, or from print or printf within a loop, you must use close() to create new instances of the command or socket. AWK does not automatically close pipes, sockets, or coprocesses when they return EOF.

The AWK versions of the printf statement and sprintf() function are similar to those of C.

For details, see https://www.gnu.org/software/gawk/manual/html_node/Printf.html.

Special File Names

When doing I/O redirection from either print or printf into a file, or via getline from a file, gawk recognizes certain special filenames internally. These filenames allow access to open file descriptors inherited from gawk's parent process (usually the shell). These file names may also be used on the command line to name data files. The filenames are:

- The standard input.

/dev/stdin The standard input.

/dev/stdout The standard output.

/dev/stderr The standard error output.

/dev/fd/n The file associated with the open file descriptor n.

The following special filenames may be used with the |& coprocess operator for creating TCP/IP network connections:

/inet/tcp/lport/rhost/rport

/inet4/tcp/lport/rhost/rport

/inet6/tcp/lport/rhost/rport

Files for a TCP/IP connection on local port lport to remote host rhost on remote port rport. Use a port of 0 to have the system pick a port. Use /inet4 to force an IPv4 connection, and /inet6 to force an IPv6 connection. Plain /inet uses the system de? fault (most likely IPv4). Usable only with the |& two-way I/O operator.

/inet/udp/lport/rhost/rport

/inet4/udp/lport/rhost/rport

/inet6/udp/lport/rhost/rport

Similar, but use UDP/IP instead of TCP/IP.

Numeric Functions

AWK has the following built-in arithmetic functions:

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

cos(expr) Return the cosine of expr, which is in radians.

exp(expr) The exponential function.

int(expr) Truncate to integer.

log(expr) The natural logarithm function.

rand() Return a random number N, between zero and one, such that 0 ? N < 1.

sin(expr) Return the sine of expr, which is in radians.

sqrt(expr) Return the square root of expr.

srand([expr]) Use expr as the new seed for the random number generator. If no expr is pro? vided, use the time of day. Return the previous seed for the random number generator.

String Functions

Gawk has the following built-in string functions; details are provided in https://www.gnu.org/software/gawk/manual/html_node/String-Functions.

asort(s [, d [, how]]) Return the number of elements in the source array s. Sort the con? tents of s using gawk's normal rules for comparing values, and re? place the indices of the sorted values s with sequential integers starting with 1. If the optional destination array d is specified, first duplicate s into d, and then sort d, leaving the indices of the source array s unchanged. The optional string how controls the direction and the comparison mode. Valid values for how are de? scribed in https://www.gnu.org/software/gawk/man? ual/html_node/String-Functions.html#String-Functions. s and d are

allowed to be the same array; this only makes sense when supplying the third argument as well.

asorti(s [, d [, how]])

Return the number of elements in the source array s. The behavior is the same as that of asort(), except that the array indices are used for sorting, not the array values. When done, the array is in? dexed numerically, and the values are those of the original indices. The original values are lost; thus provide a second array if you wish to preserve the original. The purpose of the optional string how is the same as for asort(). Here too, s and d are allowed to be the same array; this only makes sense when supplying the third argu? ment as well.

gensub(r, s, h [, t]) Search the target string t for matches of the regular expression r. If h is a string beginning with g or G, then replace all matches of r with s. Otherwise, h is a number indicating which match of r to replace. If t is not supplied, use $0 instead. Within the replace? ment text s, the sequence \n, where n is a digit from 1 to 9, may be used to indicate just the text that matched the n'th parenthesized subexpression. The sequence \0 represents the entire matched text, as does the character &. Unlike sub() and gsub(), the modified string is returned as the result of the function, and the original target string is not changed.

gsub(r, s [, t]) For each substring matching the regular expression r in the string t, substitute the string s, and return the number of substitutions. If t is not supplied, use $0. An & in the replacement text is re? placed with the text that was actually matched. Use \& to get a literal &. (This must be typed as "\\&"; see https://www.gnu.org/software/gawk/manual/html_node/Gory-De? tails.html#Gory-Details for a fuller discussion of the rules for am? persands and backslashes in the replacement text of sub(), gsub(), and gensub().)

index(s, t) Return the index of the string t in the string s, or zero if t is not present. (This implies that character indices start at one.)

length([s]) Return the length of the string s, or the length of $0 if s is not supplied. With an array argument, length() returns the number of elements in the array.

match(s, r [, a]) Return the position in s where the regular expression r occurs, or zero if r is not present, and set the values of RSTART and RLENGTH. Note that the argument order is the same as for the ~ operator: str ~ re. See https://www.gnu.org/software/gawk/man?ual/html_node/String-Functions.html#String-Functions for a descrip?tion of how the array a is filled if it is provided.

patsplit(s, a [, r [, seps]])
Split the string s into the array a and the separators array seps on the regular expression r, and return the number of fields. Element values are the portions of s that matched r. The value of seps[i] is the possibly null separator that appeared after a[i]. The value of seps[0] is the possibly null leading separator. If r is omitted, FPAT is used instead. The arrays a and seps are cleared first. Splitting behaves identically to field splitting with FPAT.

split(s, a [, r [, seps]])
Split the string s into the array a and the separators array seps on the regular expression r, and return the number of fields. If r is omitted, FS is used instead. The arrays a and seps are cleared first. seps[i] is the field separator matched by r between a[i] and a[i+1]. Splitting behaves identically to field splitting.

sprintf(fmt, expr-list) Print expr-list according to fmt, and return the resulting string.

strtonum(str) Examine str, and return its numeric value. If str begins with a leading 0, treat it as an octal number. If str begins with a lead?ing 0x or 0X, treat it as a hexadecimal number. Otherwise, assume it is a decimal number.

sub(r, s [, t]) Just like gsub(), but replace only the first matching substring. Return either zero or one.

substr(s, i [, n]) Return the at most n-character substring of s starting at i. If n is omitted, use the rest of s.

tolower(str) Return a copy of the string str, with all the uppercase characters

in str translated to their corresponding lowercase counterparts. Non-alphabetic characters are left unchanged.

toupper(str) Return a copy of the string str, with all the lowercase characters in str translated to their corresponding uppercase counterparts. Non-alphabetic characters are left unchanged.

Gawk is multibyte aware. This means that index(), length(), substr() and match() all work in terms of characters, not bytes.

Time Functions

Gawk provides the following functions for obtaining time stamps and formatting them. Details are provided in https://www.gnu.org/software/gawk/manual/html_node/Time-Functions.

mktime(datespec [, utc-flag])

Turn datespec into a time stamp of the same form as returned by systime(), and return the result. If utc-flag is present and is non-zero or non-null, the time is assumed to be in the UTC time zone; otherwise, the time is assumed to be in the local time zone. If datespec does not contain enough elements or if the resulting time is out of range, mktime() returns -1. See https://www.gnu.org/software/gawk/manual/html_node/Time-Functions.html#Time-Functions for the details of datespec.

strftime([format [, timestamp[, utc-flag]]])

Format timestamp according to the specification in format. If utc-flag is present and is non-zero or non-null, the result is in UTC, otherwise the result is in local time. The timestamp should be of the same form as returned by systime(). If timestamp is missing, the current time of day is used. If format is missing, a default format equivalent to the output of date(1) is used. The default format is available in PROCINFO["strftime"]. See the specification for the strftime() function in ISO C for the format conversions that are guaranteed to be available.

systime() Return the current time of day as the number of seconds since the Epoch (1970-01-01 00:00:00 UTC on POSIX systems).

Bit Manipulations Functions

Gawk supplies the following bit manipulation functions. They work by converting double-precision floating point values to uintmax_t integers, doing the operation, and then converting the result back to floating point. Passing negative operands to any of these functions causes a fatal error.

The functions are:

and(v1, v2 [, ...]) Return the bitwise AND of the values provided in the argument list. There must be at least two.

compl(val) Return the bitwise complement of val.

lshift(val, count) Return the value of val, shifted left by count bits.

or(v1, v2 [, ...]) Return the bitwise OR of the values provided in the argument list. There must be at least two.

rshift(val, count) Return the value of val, shifted right by count bits.

xor(v1, v2 [, ...]) Return the bitwise XOR of the values provided in the argument list. There must be at least two.

Type Functions

The following functions provide type related information about their arguments.

isarray(x) Return true if x is an array, false otherwise.

typeof(x) Return a string indicating the type of x. The string will be one of "array", "number", "regexp", "string", "strnum", "unassigned", or "undefined".

Internationalization Functions

The following functions may be used from within your AWK program for translating strings at run-time. For full details, see https://www.gnu.org/software/gawk/manual/html_node/I18N-Functions.html#I18N-Functions.

bindtextdomain(directory [, domain])

Specify the directory where gawk looks for the .gmo files, in case they will not or cannot be placed in the ?standard? locations. It returns the directory where domain is ?bound.?

The default domain is the value of TEXTDOMAIN. If directory is the null string (""), then bindtextdomain() returns the current binding for the given domain.

dcgettext(string [, domain [, category]])

Return the translation of string in text domain domain for locale category category. The default value for domain is the current value of TEXTDOMAIN. The default value for category is "LC_MESSAGES".

dcngettext(string1, string2, number [, domain [, category]])

Return the plural form used for number of the translation of string1 and string2 in text domain domain for locale category category. The default value for domain is the current value of TEXTDOMAIN. The default value for category is "LC_MESSAGES".

Boolean Valued Functions

You can create special Boolean-typed values; see the manual for how they work and why they exist.

mkbool(expression)

Based on the boolean value of expression return either a true value or a false value.

True values have numeric value one. False values have numeric value zero.

USER-DEFINED FUNCTIONS

Functions in AWK are defined as follows:

```
function name(parameter list) { statements }
```

Functions execute when they are called from within expressions in either patterns or ac? tions. Actual parameters supplied in the function call are used to instantiate the formal parameters declared in the function. Arrays are passed by reference, other variables are passed by value.

Local variables are declared as extra parameters in the parameter list. The convention is to separate local variables from real parameters by extra spaces in the parameter list. For example:

```
function  f(p, q,     a, b)   # a and b are local
{
    ...
}
/abc/     { ... ; f(1, 2) ; ... }
```

The left parenthesis in a function call is required to immediately follow the function name, without any intervening whitespace. This restriction does not apply to the built-in func? tions listed above.

Functions may call each other and may be recursive. Function parameters used as local vari? ables are initialized to the null string and the number zero upon function invocation.

Use return expr to return a value from a function. The return value is undefined if no value is provided, or if the function returns by ?falling off? the end.

Functions may be called indirectly. To do this, assign the name of the function to be called, as a string, to a variable. Then use the variable as if it were the name of a func? tion, prefixed with an @ sign, like so:

```
function myfunc()
{
```

```
    print "myfunc called"
    ...
}
{   ...
    the_func = "myfunc"
    @the_func()    # call through the_func to myfunc
    ...
}
```

If --lint has been provided, gawk warns about calls to undefined functions at parse time, instead of at run time. Calling an undefined function at run time is a fatal error.

DYNAMICALLY LOADING NEW FUNCTIONS

You can dynamically add new functions written in C or C++ to the running gawk interpreter with the @load statement. The full details are beyond the scope of this manual page; see https://www.gnu.org/software/gawk/manual/html_node/Dynamic-Extensions.html#Dynamic-Extensions.

SIGNALS

The gawk profiler accepts two signals. SIGUSR1 causes it to dump a profile and function call stack to the profile file, which is either awkprof.out, or whatever file was named with the --profile option. It then continues to run. SIGHUP causes gawk to dump the profile and function call stack and then exit.

INTERNATIONALIZATION

String constants are sequences of characters enclosed in double quotes. In non-English speaking environments, it is possible to mark strings in the AWK program as requiring translation to the local natural language. Such strings are marked in the AWK program with a leading underscore (_). For example,

```
gawk 'BEGIN { print "hello, world" }'
```

always prints hello, world. But,

```
gawk 'BEGIN { print _"hello, world" }'
```

might print bonjour, monde in France. See https://www.gnu.org/software/gawk/manual/html_node/Internationalization.html#Internationalization for the steps involved in producing and running a localizable AWK program.

GNU EXTENSIONS

Gawk has a too-large number of extensions to POSIX awk. They are described in

https://www.gnu.org/software/gawk/manual/html_node/POSIX_002fGNU.html. All the extensions can be disabled by invoking gawk with the --traditional or --posix options.

ENVIRONMENT VARIABLES

The AWKPATH environment variable can be used to provide a list of directories that gawk searches when looking for files named via the -f, --file, -i and --include options, and the @include directive. If the initial search fails, the path is searched again after appending .awk to the filename.

The AWKLIBPATH environment variable can be used to provide a list of directories that gawk searches when looking for files named via the -l and --load options.

The GAWK_PERSIST_FILE environment variable, if present, specifies a file to use as the back? ing store for persistent memory. See GAWK: Effective AWK Programming for the details.

The GAWK_READ_TIMEOUT environment variable can be used to specify a timeout in milliseconds for reading input from a terminal, pipe or two-way communication including sockets.

For connection to a remote host via socket, GAWK_SOCK_RETRIES controls the number of re? tries, and GAWK_MSEC_SLEEP the interval between retries. The interval is in milliseconds. On systems that do not support usleep(3), the value is rounded up to an integral number of seconds.

If POSIXLY_CORRECT exists in the environment, then gawk behaves exactly as if --posix had been specified on the command line. If --lint has been specified, gawk issues a warning message to this effect.

EXIT STATUS

If the exit statement is used with a value, then gawk exits with the numeric value given to it.

Otherwise, if there were no problems during execution, gawk exits with the value of the C constant EXIT_SUCCESS. This is usually zero.

If an error occurs, gawk exits with the value of the C constant EXIT_FAILURE. This is usu? ally one.

If gawk exits because of a fatal error, the exit status is 2. On non-POSIX systems, this value may be mapped to EXIT_FAILURE.

VERSION INFORMATION

This man page documents gawk, version 5.3.

AUTHORS

The original version of UNIX awk was designed and implemented by Alfred Aho, Peter Wein?

berger, and Brian Kernighan of Bell Laboratories. Ozan Yigit is the current maintainer.

Brian Kernighan occasionally dabbles in its development.

Paul Rubin and Jay Fenlason, of the Free Software Foundation, wrote gawk, to be compatible with the original version of awk distributed in Seventh Edition UNIX. John Woods con? tributed a number of bug fixes. David Trueman, with contributions from Arnold Robbins, made gawk compatible with the new version of UNIX awk. Arnold Robbins is the current maintainer.

See GAWK: Effective AWK Programming for a full list of the contributors to gawk and its doc? umentation.

See the README file in the gawk distribution for up-to-date information about maintainers and which ports are currently supported.

BUG REPORTS AND QUESTIONS

If you find a bug in gawk, please use the gawkbug(1) program to report it.

Full instructions for reporting a bug are provided in https://www.gnu.org/software/gawk/man? ual/html_node/Bugs.html. Please carefully read and follow the instructions given there.

This will make bug reporting and resolution much easier for everyone involved. Really.

On the other hand, if you have a question as to how to accomplish a particular task using awk or gawk, send an email to help-gawk@gnu.org with your request for help.

BUGS

The -F option is not necessary given the command line variable assignment feature; it re? mains only for backwards compatibility.

This manual page is too long; gawk has too many features.

SEE ALSO

egrep(1), sed(1), gawkbug(1), printf(3), and strftime(3).

The AWK Programming Language, second edition, Alfred V. Aho, Brian W. Kernighan, Peter J. Weinberger, Addison-Wesley, 2023. ISBN 9-780138-269722.

GAWK: Effective AWK Programming, Edition 5.3, shipped with the gawk source. The current version of this document is available online at https://www.gnu.org/software/gawk/manual.

The GNU gettext documentation, available online at https://www.gnu.org/software/gettext.

EXAMPLES

Print and sort the login names of all users:

```
BEGIN     { FS = ":" }
      { print $1 | "sort" }
```

Count lines in a file:

```
    { nlines++ }

END  { print nlines }
```

Precede each line by its number in the file:

```
{ print FNR, $0 }
```

Concatenate and line number (a variation on a theme):

```
{ print NR, $0 }
```

Run an external command for particular lines of data:

```
tail -f access_log |
awk '/myhome.html/ { system("nmap " $1 ">> logdir/myhome.html") }'
```

COPYING PERMISSIONS