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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'awk.1' command

$ 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 ...

DESCRIPTION

Gawk is the GNU Project's implementation of the AWK programming lan? guage. 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.

When gawk is invoked with the --profile option, it starts gathering profiling statistics from the execution of the program. Gawk runs more slowly in this mode, and automatically produces an execution profile in the file awkprof.out when done. See the --profile option, below.

Gawk also has an integrated debugger. An interactive debugging session can be started by supplying the --debug option to the command line. In this mode of execution, gawk loads the AWK source code and then prompts

for debugging commands. Gawk can only debug AWK program source pro? vided with the -f and --include options. The debugger is documented in GAWK: Effective AWK Programming.

OPTION FORMAT

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

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

Additionally, every long option has a corresponding short option, so that the option's functionality 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 (or --file) 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 prede? fined 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. In other words, don't pay any attention to the locale information when attempt? ing to process strings as multibyte characters. The --posix op? tion 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. See GNU EXTENSIONS, below, for more information.

-C

--copyright

Print the short version of the GNU copyright information message on the standard output and exit successfully.

-d[file]

--dump-variables[=file]

Print a sorted list of global variables, their types and final values to file. If no file is provided, gawk uses a file named awkvars.out in the current directory.

Having a list of all the global variables is a good way to look for typographical errors in your programs. You would also use this option if you have a large program with a lot of functions, and you want to be sure that your functions don't inadvertently use global variables that you meant to be local. (This is a particularly easy mistake to make with simple variable names like i, j, and so on.)

-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 com?

mands for the debugger to execute non-interactively.

-e program-text

--source program-text

Use program-text as AWK program source code. This option allows

the easy intermixing of library functions (used via the -f and

--include options) with source code entered on the command line.

It is intended primarily for medium to large AWK programs used

in shell scripts. 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 is option is the last one pro?

cessed. 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 (Porta?

ble Object Template) format file on standard output with entries

for all localizable strings in the program. The program itself

is not executed. See the GNU gettext distribution for more in?

formation on .pot files.

-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 elimi? nated), 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" statement.

-l lib

--load lib

Load a gawk extension from the shared library lib. This searches for the library using the AWKLIBPATH environment vari? able. If the initial search fails, another attempt 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-porta? ble to other AWK implementations. With an optional argument of fatal, lint warnings become fatal errors. This may be drastic, but its use will certainly encourage the development of cleaner AWK programs. With an optional argument of invalid, only warn? ings about things that are actually invalid are issued. (This is not fully implemented yet.) With an optional argument of no- ext, warnings about gawk extensions are disabled.

-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.)

-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. Although the POSIX standard requires this behavior, and gawk does so when --posix is in effect, the de? fault is to follow traditional behavior and use a period as the decimal point, even in locales where the period is not the deci? mal point character. This option overrides the default behav? ior, without the full draconian strictness of the --posix op? tion.

-o[file]

--pretty-print[=file]

Output a pretty printed version of the program to file. If no file is provided, gawk uses a file named awkprof.out in the cur? rent directory. This option implies --no-optimize.

-O

--optimize

Enable gawk's default optimizations upon the internal represen? tation of the program. Currently, this just includes simple constant folding. 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. The profile contains execu? tion counts of each statement in the program in the left margin and function call counts for each user-defined function. This option implies --no-optimize.

-P

--posix

This turns on compatibility mode, with the following additional restrictions:

? \x escape sequences are not recognized.

? You cannot continue lines after ? and :.

? The synonym func for the keyword function is not recognized.

? The operators ** and **= cannot be used in place of ^ and ^=.

-r

--re-interval

Enable the use of interval expressions in regular expression matching (see Regular Expressions, below). Interval expressions were not traditionally available in the AWK language. The POSIX standard added them, to make awk and egrep consistent with each other. They are enabled by default, but this option remains for use together with --traditional.

-s

--no-optimize

Disable gawk's default optimizations upon the internal represen? tation 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. This effectively blocks a script from accessing local resources, except for the files specified on the command line.

-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 mainly for knowing if the current copy of gawk on your system is up to date with respect to whatever the Free Software Foundation is distributing. This is also useful when reporting bugs. (Per the GNU Coding Stan? dards, these options cause an immediate, successful exit.)

-- Signal the end of options. This is useful to allow further argu?
ments to the AWK program itself to start with a ?-?. This pro?
vides consistency with the argument parsing convention used by
most other POSIX programs.

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. This is particularly useful for running AWK programs via the #! executable interpreter mechanism.

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

AWK PROGRAM EXECUTION

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 speci? fied, 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 to? gether. This is useful for building libraries of AWK functions, with? out having to include them in each new AWK program that uses them. It also provides the ability to mix library functions with command line programs.

In addition, lines beginning with @include may be used to include other source files into your program, making library use even easier. 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 ?/? char? acter, 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 specified via the -v option are performed. Next, gawk com? piles 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 assignment. The variable var will be assigned the value val. (This happens after any BEGIN rule(s) have been run.) Command line variable assignment is most useful for dynamically assigning val? ues to the variables AWK uses to control how input is broken into fields and records. It is also useful for controlling state if multi? ple passes are needed over a single data file.

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 as? sociated code before processing the contents of the file. Similarly, gawk executes the code associated with ENDFILE after processing the file.

For each record in the input, gawk tests to see if it matches any pat? tern in the AWK program. 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.

Starting with version 4.0 of gawk, a directory on the command line pro? duces a warning, but is otherwise skipped. If either of the --posix or --traditional options is given, then gawk reverts to treating directo? ries 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 dimen? sional arrays; arrays with multiple dimensions may be simulated. Gawk provides true arrays of arrays; see Arrays, below. 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 separated by assigning values to the built-in variable RS. If RS is any single character, that character separates records. Otherwise, RS is a regular expression. Text in the input that matches this regular expression separates the record. However, in compatibil? ity mode, only the first character of its string value is used for sep? arating records. If RS is set to the null string, then records are separated by empty lines. When RS is set to the null string, the new? line character always acts as a field separator, in addition to what? ever value FS may have.

Fields

As each input record is read, gawk splits the record into fields, using the value of the FS variable as the field separator. If FS is a single

character, fields are separated by that character. If FS is the null string, then each individual character becomes a separate field. Oth? erwise, FS is expected to be a full regular expression. In the special case that FS is a single space, fields are separated by runs of spaces and/or tabs and/or newlines. NOTE: The value of IGNORECASE (see below) also affects how fields are split when FS is a regular expression, and how records are separated when RS is a regular expression.

If the FIELDWIDTHS variable is set to a space-separated list of num? bers, each field is expected to have fixed width, and gawk splits up the record using the specified widths. Each field width may optionally be preceded by a colon-separated value specifying the number of charac? ters to skip before the field starts. The value of FS is ignored. As? signing a new value to FS or FPAT overrides the use of FIELDWIDTHS.

Similarly, if the FPAT variable is set to a string representing a regu? lar expression, each field is made up of text that matches that regular expression. In this case, the regular expression describes the fields themselves, instead of the text that separates the fields. Assigning a new value to FS or FIELDWIDTHS overrides the use of FPAT.

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. Fields need not be referenced by constants:

```
n = 5
print $n
```

prints the fifth field in the input record.

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. However, assigning to a non-existent field (e.g., $(NF+2) = 5) increases the value of NF, creates any intervening fields with the null string as their values, and causes the value of $0 to be recom? puted, with the fields being separated by the value of OFS. References to negative 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 separated by the value of OFS.

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

Built-in Variables

Gawk's built-in variables are:

ARGC The number of command line arguments (does not include options to gawk, or the program source).

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. Dynamically changing the contents of ARGV can control the files used for data.

BINMODE On non-POSIX systems, specifies use of ?binary? mode for all file I/O. Numeric values of 1, 2, or 3, specify that input files, output files, or all files, respectively, should use binary I/O. String values of "r", or "w" specify that input files, or output files, respectively, should use binary I/O. String values of "rw" or "wr" specify that all files should use binary I/O. Any other string value is treated as "rw", but generates a warning message.

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 (e.g., ENVIRON["HOME"] might be "/home/arnold").

In POSIX mode, changing this array does not affect the environment seen by programs which gawk spawns via redirection or the system() function. Otherwise, gawk updates its real environment so that programs it spawns see the changes.

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. If the string in ERRNO corresponds to a system error in the errno(3) variable, then the numeric value can be found in PROCINFO["errno"]. For non-system errors, PROCINFO["er? rno"] will be zero.

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 separa? tor. Each field width may optionally be preceded by a colon-separated value specifying the number of characters to skip before the field starts. See Fields, above.

FILENAME The name of the current input file. If no files are speci? fied on the command line, the value of FILENAME is ?-?. However, FILENAME is undefined inside the BEGIN rule (un? less 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 expression, instead of using the value of FS as the field separator. See Fields, above.

FS The input field separator, a space by default. See Fields, above.

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. If IGNORECASE has a non-zero value, then string comparisons and pattern matching in rules, field splitting with FS and FPAT, record separating with RS, regular expression matching with ~ and !~, and the gen?

sub(), gsub(), index(), match(), patsplit(), split(), and sub() built-in functions all ignore case when doing regular expression operations. NOTE: Array subscripting is not af? fected. However, the asort() and asorti() functions are affected.

Thus, if IGNORECASE is not equal to zero, /aB/ matches all of the strings "ab", "aB", "Ab", and "AB". As with all AWK variables, the initial value of IGNORECASE is zero, so all regular expression and string operations are normally case-sensitive.

LINT Provides dynamic control of the --lint option from within an AWK program. When true, gawk prints lint warnings. When false, it does not. The values allowed for the --lint op? tion may also be assigned to LINT, with the same effects. Any other true value just prints warnings.

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 default.

PROCINFO The elements of this array provide access to information about the running AWK program. On some systems, there may be elements in the array, "group1" through "groupn" for some n, which is the number of supplementary groups that the process has. Use the in operator to test for these el? ements. The following elements are guaranteed to be avail? able:

PROCINFO["argv"] The command line arguments as received by gawk at the C-language level. The subscripts start from zero.

PROCINFO["egid"] The value of the getegid(2) system

call.

PROCINFO["errno"] The value of errno(3) when ERRNO is set to the associated error message.

PROCINFO["euid"] The value of the geteuid(2) system call.

PROCINFO["FS"] "FS" if field splitting with FS is in effect, "FPAT" if field splitting with FPAT is in effect, "FIELDWIDTHS" if field splitting with FIELDWIDTHS is in effect, or "API" if API input parser field splitting is in effect.

PROCINFO["gid"] The value of the getgid(2) system call.

PROCINFO["identifiers"]

A subarray, indexed by the names of all identifiers used in the text of the AWK program. The values indicate what gawk knows about the identifiers after it has finished parsing the pro? gram; they are not updated while the program runs. For each identifier, the value of the element is one of the following:

"array" The identifier is an ar? ray.

"builtin" The identifier is a built- in function.

"extension" The identifier is an ex? tension function loaded via @load or --load.

"scalar" The identifier is a scalar.

"untyped" The identifier is untyped

(could be used as a scalar
or array, gawk doesn't
know yet).

"user" The identifier is a user-
defined function.

PROCINFO["pgrpid"] The value of the getpgrp(2) system
call.

PROCINFO["pid"] The value of the getpid(2) system
call.

PROCINFO["platform"] A string indicating the platform for
which gawk was compiled. It is one
of:

"djgpp", "mingw"
Microsoft Windows, using either
DJGPP, or MinGW, respectively.

"os2" OS/2.

"posix"
GNU/Linux, Cygwin, Mac OS X,
and legacy Unix systems.

"vms" OpenVMS or Vax/VMS.

PROCINFO["ppid"] The value of the getppid(2) system
call.

PROCINFO["strftime"] The default time format string for
strftime(). Changing its value af?
fects how strftime() formats time val?
ues when called with no arguments.

PROCINFO["uid"] The value of the getuid(2) system
call.

PROCINFO["version"] The version of gawk.

The following elements are present if loading dynamic ex?
tensions is available:

PROCINFO["api_major"]
The major version of the extension API.

PROCINFO["api_minor"]

The minor version of the extension API.

The following elements are available if MPFR support is compiled into gawk:

PROCINFO["gmp_version"]

The version of the GNU GMP library used for arbi? trary precision number support in gawk.

PROCINFO["mpfr_version"]

The version of the GNU MPFR library used for arbi? trary precision number support in gawk.

PROCINFO["prec_max"]

The maximum precision supported by the GNU MPFR li? brary for arbitrary precision floating-point num? bers.

PROCINFO["prec_min"]

The minimum precision allowed by the GNU MPFR li? brary for arbitrary precision floating-point num? bers.

The following elements may set by a program to change gawk's behavior:

PROCINFO["NONFATAL"]

If this exists, then I/O errors for all redirections become nonfatal.

PROCINFO["name", "NONFATAL"]

Make I/O errors for name be nonfatal.

PROCINFO["command", "pty"]

Use a pseudo-tty for two-way communication with com? mand instead of setting up two one-way pipes.

PROCINFO["input", "READ_TIMEOUT"]

The timeout in milliseconds for reading data from input, where input is a redirection string or a filename. A value of zero or less than zero means no timeout.

PROCINFO["input", "RETRY"]

If an I/O error that may be retried occurs when reading data from input, and this array entry ex? ists, then getline returns -2 instead of following the default behavior of returning -1 and configuring input to return no further data. An I/O error that may be retried is one where errno(3) has the value EAGAIN, EWOULDBLOCK, EINTR, or ETIMEDOUT. This may be useful in conjunction with PROCINFO["input", "READ_TIMEOUT"] or in situations where a file de? scriptor has been configured to behave in a non-blocking fashion.

PROCINFO["sorted_in"]

If this element exists in PROCINFO, then its value controls the order in which array elements are tra? versed in for loops. Supported values are "@ind_str_asc", "@ind_num_asc", "@val_type_asc", "@val_str_asc", "@val_num_asc", "@ind_str_desc", "@ind_num_desc", "@val_type_desc", "@val_str_desc", "@val_num_desc", and "@unsorted". The value can also be the name (as a string) of any comparison function defined as follows:

function cmp_func(i1, v1, i2, v2)

where i1 and i2 are the indices, and v1 and v2 are the corresponding values of the two elements being compared. It should return a number less than, equal to, or greater than 0, depending on how the elements of the array are to be ordered.

ROUNDMODE The rounding mode to use for arbitrary precision arithmetic on numbers, by default "N" (IEEE-754 roundTiesToEven mode). The accepted values are:

"A" or "a"

for rounding away from zero. These are only avail?

able if your version of the GNU MPFR library sup?
ports rounding away from zero.

"D" or "d" for roundTowardNegative.

"N" or "n" for roundTiesToEven.

"U" or "u" for roundTowardPositive.

"Z" or "z" for roundTowardZero.

RS The input record separator, by default a newline.

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

RSTART The index of the first character matched by match(); 0 if no match. (This implies that character indices start at one.)

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 de?
fined global variables and arrays in the program. The ar?
ray may be used for indirect access to read or write the value of a variable:

```
foo = 5
SYMTAB["foo"] = 4
print foo    # prints 4
```

The typeof() function may be used to test if an element in SYMTAB is an array. 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 local?
ized translations for the program's strings.

Arrays

Arrays are subscripted with an expression between square brackets ([and]). If the expression 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 ar? rays. For example:

```
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 associa? tive, i.e., indexed by string values.

The special operator in may be used to test if an array has an index consisting of a particular 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 array. 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 ``rectangular'' as in C or C++. For example:

```
a[1] = 5
a[2][1] = 6
a[2][2] = 7
```

NOTE: You may need to tell gawk that an array element is really a sub? array in order to use it where gawk expects an array (such as in the second argument to split()). You can do this by creating an element in the subarray and then deleting it with the delete statement.

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 dou?

ble 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'' namespace; the default namespace is awk. However, simple identifiers consisting solely of uppercase letters are forced into the awk namespace, even if the current namespace is different.

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

The standard predefined builtin function names may not be used as name? space 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 GAWK: Effective AWK Programming.

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 vari? able is interpreted depends upon its context. If used in a numeric ex? pression, 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 accom? plished 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 nu? meric value of the variable as the argument. However, even though all numbers in AWK are floating-point, integral values are always converted as integers. Thus, given

```
CONVFMT = "%2.2f"
a = 12
b = a ""
```

the variable b has a string value of "12" and not "12.00".

NOTE: When operating in POSIX mode (such as with the --posix option), beware that locale settings may interfere with the way decimal numbers

are treated: the decimal separator of the numbers you are feeding to gawk must conform to what your locale would expect, be it a comma (,) or a period (.).

Gawk performs comparisons as follows: If two variables are numeric, they are compared numerically. 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 converted 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 constants. 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 nu? meric 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. These are:

\\ A literal backslash.

\a The ?alert? character; usually the ASCII BEL character.

\b Backspace.

\f Form-feed.

\n Newline.

\r Carriage return.

\t Horizontal tab.

\v Vertical tab.

\xhex digits

The character represented by the string of hexadecimal digits fol?

lowing the \x. Up to two following hexadecimal digits are consid? ered part of the escape sequence. E.g., "\x1B" is the ASCII ESC (escape) character.

\ddd The character represented by the 1-, 2-, or 3-digit sequence of octal digits. E.g., "\033" is the ASCII ESC (escape) character.

\c The literal character c.

In compatibility mode, the characters represented by octal and hexadec? imal escape sequences are treated literally when used in regular ex? pression constants. Thus, /a\52b/ is equivalent to /a*b/.

Regexp Constants

A regular expression constant is a sequence of characters enclosed be? tween forward slashes (like /value/). Regular expression matching is described more fully below; see Regular Expressions.

The escape sequences described earlier may also be used inside constant regular expressions (e.g., /[\t\f\n\r\v]/ matches whitespace charac? ters).

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 regu? lar 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 ignored. 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 state? ments 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 statement is executed). BEGIN and END patterns cannot be com? bined with other patterns in pattern expressions. BEGIN and END pat? terns 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 BEGIN?

FILE 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 ex? pressions are the same as those in egrep(1), and are summarized below.

A relational expression may use any of the operators defined below in the section on actions. 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 primitive pattern expressions. As in most languages, parentheses may be used to change the order of evaluation.

The ?: operator is like the same operator in C. If the first pattern is true then the pattern used for testing is the second pattern, other? wise 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 input records starting with a record that matches pat? tern1, and continuing until a record that matches pattern2, inclusive. It does not combine with any other sort of pattern expression.

Regular Expressions

Regular expressions are the extended kind found in egrep. They are composed of characters as follows:

c Matches the non-metacharacter c.

\c Matches the literal character c.

. Matches any character including newline.

^ Matches the beginning of a string.

$ Matches the end of a string.

[abc...] A character list: matches any of the characters abc.... You may include a range of characters by separating them with a

dash. To include a literal dash in the list, put it first or last.

[^abc...] A negated character list: matches any character except abc....

r1|r2 Alternation: matches either r1 or r2.

r1r2 Concatenation: matches r1, and then r2.

r+ Matches one or more r's.

r* Matches zero or more r's.

r? Matches zero or one r's.

(r) Grouping: matches r.

r{n}

r{n,}

r{n,m} One or two numbers inside braces denote an interval expression. If there is one number in the braces, the preceding regular expression r is repeated n times. If there are two numbers separated by a comma, r is repeated n to m times. If there is one number followed by a comma, then r is repeated at least n times.

\y Matches the empty string at either the beginning or the end of a word.

\B Matches the empty string within a word.

\< Matches the empty string at the beginning of a word.

\> Matches the empty string at the end of a word.

\s Matches any whitespace character.

\S Matches any nonwhitespace character.

\w Matches any word-constituent character (letter, digit, or underscore).

\W Matches any character that is not word-constituent.

\` Matches the empty string at the beginning of a buffer (string).

\' Matches the empty string at the end of a buffer.

The escape sequences that are valid in string constants (see String Constants) are also valid in regular expressions.

Character classes are a feature introduced in the POSIX standard. A character class is a special notation for describing lists of charac? ters that have a specific attribute, but where the actual characters themselves can vary from country to country and/or from character set to character set. For example, the notion of what is an alphabetic character differs in the USA and in France.

A character class is only valid in a regular expression inside the brackets of a character list. Character classes consist of [:, a key? word denoting the class, and :]. The character classes defined by the POSIX standard are:

[:alnum:] Alphanumeric characters.

[:alpha:] Alphabetic characters.

[:blank:] Space or tab characters.

[:cntrl:] Control characters.

[:digit:] Numeric characters.

[:graph:] Characters that are both printable and visible. (A space is printable, but not visible, while an a is both.)

[:lower:] Lowercase alphabetic characters.

[:print:] Printable characters (characters that are not control char? acters.)

[:punct:] Punctuation characters (characters that are not letter, dig? its, control characters, or space characters).

[:space:] Space characters (such as space, tab, and formfeed, to name a few).

[:upper:] Uppercase alphabetic characters.

[:xdigit:] Characters that are hexadecimal digits.

For example, before the POSIX standard, to match alphanumeric charac? ters, you would have had to write /[A-Za-z0-9]/. If your character set had other alphabetic characters in it, this would not match them, and if your character set collated differently from ASCII, this might not even match the ASCII alphanumeric characters. With the POSIX character classes, you can write /[[:alnum:]]/, and this matches the alphabetic and numeric characters in your character set, no matter what it is.

Two additional special sequences can appear in character lists. These apply to non-ASCII character sets, which can have single symbols (called collating elements) that are represented with more than one character, as well as several characters that are equivalent for col? lating, or sorting, purposes. (E.g., in French, a plain ?e? and a grave-accented ?`? are equivalent.)

Collating Symbols

A collating symbol is a multi-character collating element en? closed in [. and .]. For example, if ch is a collating ele? ment, then [[.ch.]] is a regular expression that matches this collating element, while [ch] is a regular expression that matches either c or h.

Equivalence Classes

An equivalence class is a locale-specific name for a list of characters that are equivalent. The name is enclosed in [= and =]. For example, the name e might be used to represent all of ?e?, ???, and ?`?. In this case, [[=e=]] is a regular expres? sion that matches any of e, ?, or `.

These features are very valuable in non-English speaking locales. The library functions that gawk uses for regular expression matching cur? rently only recognize POSIX character classes; they do not recognize collating symbols or equivalence classes.

The \y, \B, \<, \>, \s, \S, \w, \W, \`, and \' operators are specific to gawk; they are extensions based on facilities in the GNU regular ex? pression libraries.

The various command line options control how gawk interprets characters in regular expressions.

No options

In the default case, gawk provides all the facilities of POSIX regular expressions and the GNU regular expression operators de? scribed above.

--posix

Only POSIX regular expressions are supported, the GNU operators

are not special. (E.g., \w matches a literal w).

--traditional

Traditional UNIX awk regular expressions are matched. The GNU operators are not special, and interval expressions are not available. Characters described by octal and hexadecimal escape sequences are treated literally, even if they represent regular expression metacharacters.

--re-interval

Allow interval expressions in regular expressions, even if --traditional has been provided.

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 in? put/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 (** may also be used, and **= for the as? signment operator).

+ - ! 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. NOTE: Do not use a constant regular expression (/foo/) on the left-hand side of a ~ or !~. Only use one on the right-hand side. The expression /foo/ ~ exp has the same meaning as (($0 ~

/foo/) ~ exp). This is usually not what you want.

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 expres? sion is expr2, otherwise it is expr3. Only one of expr2 and expr3 is evaluated.

= += -= *= /= %= ^= Assignment. Both absolute assignment (var = value) and op? erator-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 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 ei? ther into $0 or var, as above, and RT. Copro? cesses are a gawk extension. (The command can also be a socket. See the subsection Special File Names, below.)

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 program. 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 termi? nated 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. See The printf Statement, be? low.

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 GAWK: Effective AWK Programming for the full details on the exit status.

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.

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, below.)

The getline command 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 re? turned instead of -1, and further calls to getline may be attempted. 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 printf Statement

The AWK versions of the printf statement and sprintf() function (see below) accept the following conversion specification formats:

%a, %A A floating point number of the form [-]0xh.hhhhp+-dd (C99 hexa? decimal floating point format). For %A, uppercase letters are used instead of lowercase ones.

%c A single character. If the argument used for %c is numeric, it is treated as a character and printed. Otherwise, the argument is assumed to be a string, and the only first character of that string is printed.

%d, %i A decimal number (the integer part).

%e, %E A floating point number of the form [-]d.dddddde[+-]dd. The %E format uses E instead of e.

%f, %F A floating point number of the form [-]ddd.dddddd. If the sys? tem library supports it, %F is available as well. This is like %f, but uses capital letters for special ?not a number? and ?infinity? values. If %F is not available, gawk uses %f.

%g, %G Use %e or %f conversion, whichever is shorter, with nonsignifi? cant zeros suppressed. The %G format uses %E instead of %e.

%o An unsigned octal number (also an integer).

%u An unsigned decimal number (again, an integer).

%s A character string.

%x, %X An unsigned hexadecimal number (an integer). The %X format uses ABCDEF instead of abcdef.

%% A single % character; no argument is converted.

Optional, additional parameters may lie between the % and the control letter:

count$ Use the count'th argument at this point in the formatting. This is called a positional specifier and is intended primarily for use in translated versions of format strings, not in the origi? nal text of an AWK program. It is a gawk extension.

- The expression should be left-justified within its field.

space For numeric conversions, prefix positive values with a space, and negative values with a minus sign.

+ The plus sign, used before the width modifier (see below), says to always supply a sign for numeric conversions, even if the data to be formatted is positive. The + overrides the space modifier.

Use an ?alternate form? for certain control letters. For %o, supply a leading zero. For %x, and %X, supply a leading 0x or 0X for a nonzero result. For %e, %E, %f and %F, the result al? ways contains a decimal point. For %g, and %G, trailing zeros are not removed from the result.

0 A leading 0 (zero) acts as a flag, indicating that output should be padded with zeroes instead of spaces. This applies only to the numeric output formats. This flag only has an effect when the field width is wider than the value to be printed.

' A single quote character instructs gawk to insert the locale's thousands-separator character into decimal numbers, and to also use the locale's decimal point character with floating point formats. This requires correct locale support in the C library and in the definition of the current locale.

width The field should be padded to this width. The field is normally padded with spaces. With the 0 flag, it is padded with zeroes.

.prec A number that specifies the precision to use when printing. For the %e, %E, %f and %F, formats, this specifies the number of digits you want printed to the right of the decimal point. For the %g, and %G formats, it specifies the maximum number of sig? nificant digits. For the %d, %i, %o, %u, %x, and %X formats, it specifies the minimum number of digits to print. For the %s format, it specifies the maximum number of characters from the string that should be printed.

The dynamic width and prec capabilities of the ISO C printf() routines are supported. A * in place of either the width or prec specifications causes their values to be taken from the argument list to printf or

sprintf(). To use a positional specifier with a dynamic width or pre?
cision, supply the count$ after the * in the format string. For exam?
ple, "%3$*2$.*1$s".

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 in? ternally. These filenames allow access to open file descriptors inher? ited 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.

These are particularly useful for error messages. For example:

```
print "You blew it!" > "/dev/stderr"
```

whereas you would otherwise have to use

```
print "You blew it!" | "cat 1>&2"
```

The following special filenames may be used with the |& coprocess oper? ator 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 provided, use the time of day. Return the previous seed for the random number generator.

String Functions

Gawk has the following built-in string functions:

asort(s [, d [, how]]) Return the number of elements in the source ar? ray s. Sort the contents of s using gawk's normal rules for comparing values, and replace the indices of the sorted values s with sequen? tial integers starting with 1. If the optional destination array d is specified, first dupli? cate s into d, and then sort d, leaving the in? dices of the source array s unchanged. The op? tional string how controls the direction and the comparison mode. Valid values for how are any of the strings valid for PROCINFO["sorted_in"]. It can also be the name of a user-defined comparison function as de? scribed in PROCINFO["sorted_in"].

asorti(s [, d [, how]]) Return the number of elements in the source ar?

ray 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 indexed numerically, and the val? ues 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 described previously for asort().

gensub(r, s, h [, t]) Search the target string t for matches of the regular expression r. If h is a string begin? ning 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 sup? plied, use $0 instead. Within the replacement 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 subex? pression. The sequence \0 represents the en? tire 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 expres? sion 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 re? placement text is replaced with the text that was actually matched. Use \& to get a literal &. (This must be typed as "\\&"; see GAWK: Ef? fective AWK Programming for a fuller discussion of the rules for ampersands and backslashes in the replacement text of sub(), gsub(), and gen?

sub().)

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.) It is a fatal error to use a regexp constant for t.

length([s]) Return the length of the string s, or the length of $0 if s is not supplied. As a non-standard extension, 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 ex? pression 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. If array a is provided, a is cleared and then elements 1 through n are filled with the portions of s that match the corresponding parenthesized subexpression in r. The zero'th element of a contains the portion of s matched by the entire regular expression r. Subscripts a[n, "start"], and a[n, "length"] provide the starting index in the string and length respectively, of each match? ing substring.

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 sepa? rator 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 ar?

rays a and seps are cleared first. Splitting behaves identically to field splitting with FPAT, described above.

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]. If r is a single space, then leading whitespace in s goes into the extra array element seps[0] and trailing whitespace goes into the extra ar? ray element seps[n], where n is the return value of split(s, a, r, seps). Splitting be? haves identically to field splitting, described above. In particular, if r is a single-charac? ter string, that string acts as the separator, even if it happens to be a regular expression metacharacter.

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 oc? tal number. If str begins with a leading 0x or 0X, treat it as a hexadecimal number. Other? wise, 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-al? phabetic 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-al? phabetic 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

Since one of the primary uses of AWK programs is processing log files that contain time stamp information, gawk provides the following func? tions for obtaining time stamps and formatting them.

mktime(datespec [, utc-flag])

Turn datespec into a time stamp of the same form as returned by systime(), and return the result. The datespec is a string of the form YYYY MM DD HH MM SS[DST]. The contents of the string are six or seven numbers representing respec? tively the full year including century, the month from 1 to 12, the day of the month from 1 to 31, the hour of the day from 0 to 23, the minute from 0 to 59, the second from 0 to 60, and an optional daylight saving flag. The values of these numbers need not be within the ranges specified; for example, an hour of -1 means 1 hour before midnight. The origin-zero Gregorian calendar is assumed, with year 0 pre? ceding year 1 and year -1 preceding year 0. 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 the DST daylight saving flag is positive, the time is assumed to be daylight saving time; if zero, the time is assumed to be standard time; and if nega? tive (the default), mktime() attempts to determine whether daylight saving time is in effect for the specified time. If

datespec does not contain enough elements or if the resulting time is out of range, mktime() returns -1.

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.

NOTE: 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. This func? tion is mainly for use with the elements of multidimensional arrays and with function parameters.

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 GAWK: Effective AWK Programming.

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'' loca? tions (e.g., during testing). 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 lo? cale category category. The default value for domain is the current value of TEXTDOMAIN. The default value for category is "LC_MESSAGES".

If you supply a value for category, it must be a string equal to one of the known locale categories described in GAWK: Effective AWK Programming. You must also supply a text domain. Use TEXTDOMAIN if you want to use the current domain.

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". If you supply a value for category, it must be a string equal to one of the known locale categories described in GAWK: Effective AWK Programming. You must also supply a text domain. Use TEXTDOMAIN if you want to use the current domain.

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 actions. 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.

Since functions were not originally part of the AWK language, the provision for local variables is rather clumsy: They 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 avoids a syntactic ambiguity with the concatenation operator. This restriction does not apply to the built-in functions listed above.

Functions may call each other and may be recursive. Function parameters used as local variables 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 ?fall?
ing off? the end.

As a gawk extension, functions may be called indirectly. To do this, assign the name of the function to be called, as a string, to a vari? able. Then use the variable as if it were the name of a function, pre? fixed with an @ sign, like so:

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

As of version 4.1.2, this works with user-defined functions, built-in functions, and extension functions.

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

The word func may be used in place of function, although this is depre? cated.

DYNAMICALLY LOADING NEW FUNCTIONS

You can dynamically add new functions written in C or C++ to the run? ning gawk interpreter with the @load statement. The full details are beyond the scope of this manual page; see GAWK: Effective AWK Program? ming.

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.

There are several steps involved in producing and running a localizable AWK program.

1. Add a BEGIN action to assign a value to the TEXTDOMAIN variable to set the text domain to a name associated with your program:

   ```
   BEGIN { TEXTDOMAIN = "myprog" }
   ```

   This allows gawk to find the .gmo file associated with your pro? gram. Without this step, gawk uses the messages text domain, which likely does not contain translations for your program.

2. Mark all strings that should be translated with leading under? scores.

3. If necessary, use the dcgettext() and/or bindtextdomain() functions in your program, as appropriate.

4. Run gawk --gen-pot -f myprog.awk > myprog.pot to generate a .pot file for your program.

5. Provide appropriate translations, and build and install the corre? sponding .gmo files.

The internationalization features are described in full detail in GAWK: Effective AWK Programming.

POSIX COMPATIBILITY

A primary goal for gawk is compatibility with the POSIX standard, as well as with the latest version of Brian Kernighan's awk. To this end, gawk incorporates the following user visible features which are not de?

scribed in the AWK book, but are part of the Brian Kernighan's version of awk, and are in the POSIX standard.

The book indicates that command line variable assignment happens when awk would otherwise open the argument as a file, which is after the BE? GIN rule is executed. However, in earlier implementations, when such an assignment appeared before any file names, the assignment would hap? pen before the BEGIN rule was run. Applications came to depend on this ?feature.? When awk was changed to match its documentation, the -v op? tion for assigning variables before program execution was added to ac? commodate applications that depended upon the old behavior. (This fea? ture was agreed upon by both the Bell Laboratories developers and the GNU developers.)

When processing arguments, gawk uses the special option ?--? to signal the end of arguments. In compatibility mode, it warns about but other? wise ignores undefined options. In normal operation, such arguments are passed on to the AWK program for it to process.

The AWK book does not define the return value of srand(). The POSIX standard has it return the seed it was using, to allow keeping track of random number sequences. Therefore srand() in gawk also returns its current seed.

Other features are: The use of multiple -f options (from MKS awk); the ENVIRON array; the \a, and \v escape sequences (done originally in gawk and fed back into the Bell Laboratories version); the tolower() and toupper() built-in functions (from the Bell Laboratories version); and the ISO C conversion specifications in printf (done first in the Bell Laboratories version).

HISTORICAL FEATURES

There is one feature of historical AWK implementations that gawk sup? ports: It is possible to call the length() built-in function not only with no argument, but even without parentheses! Thus,

```
a = length     # Holy Algol 60, Batman!
```

is the same as either of

```
a = length()
```

a = length($0)

Using this feature is poor practice, and gawk issues a warning about its use if --lint is specified on the command line.

GNU EXTENSIONS

Gawk has a too-large number of extensions to POSIX awk. They are described in this section. All the extensions described here can be disabled by invoking gawk with the --traditional or --posix options.

The following features of gawk are not available in POSIX awk.

? No path search is performed for files named via the -f option. Therefore the AWKPATH environment variable is not special.

? There is no facility for doing file inclusion (gawk's @include mechanism).

? There is no facility for dynamically adding new functions written in C (gawk's @load mechanism).

? The \x escape sequence.

? The ability to continue lines after ? and :.

? Octal and hexadecimal constants in AWK programs.

? The ARGIND, BINMODE, ERRNO, LINT, PREC, ROUNDMODE, RT and TEXTDOMAIN variables are not special.

? The IGNORECASE variable and its side-effects are not available.

? The FIELDWIDTHS variable and fixed-width field splitting.

? The FPAT variable and field splitting based on field values.

? The FUNCTAB, SYMTAB, and PROCINFO arrays are not available.

? The use of RS as a regular expression.

? The special file names available for I/O redirection are not recognized.

? The |& operator for creating coprocesses.

? The BEGINFILE and ENDFILE special patterns are not available.

? The ability to split out individual characters using the null string as the value of FS, and as the third argument to split().

? An optional fourth argument to split() to receive the separator texts.

? The optional second argument to the close() function.

? The optional third argument to the match() function.

? The ability to use positional specifiers with printf and sprintf().

? The ability to pass an array to length().

? The and(), asort(), asorti(), bindtextdomain(), compl(), dcgettext(), dcngettext(), gensub(), lshift(), mktime(), or(), patsplit(), rshift(), strftime(), strtonum(), systime() and xor() functions.

? Localizable strings.

? Non-fatal I/O.

? Retryable I/O.

The AWK book does not define the return value of the close() function. Gawk's close() returns the value from fclose(3), or pclose(3), when closing an output file or pipe, respectively. It returns the process's exit status when closing an input pipe. The return value is -1 if the named file, pipe or coprocess was not opened with a redirection.

When gawk is invoked with the --traditional option, if the fs argument to the -F option is ?t?, then FS is set to the tab character. Note that typing gawk -F\t ... simply causes the shell to quote the ?t,? and does not pass ?\t? to the -F option. Since this is a rather ugly special case, it is not the default behavior. This behavior also does not occur if --posix has been specified. To really get a tab character as the field separator, it is best to use single quotes: gawk -F'\t' ....

ENVIRONMENT VARIABLES

The AWKPATH environment variable can be used to provide a list of di? rectories 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_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 retries, and GAWK_MSEC_SLEEP the interval between re? tries. 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 usually 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 5.1.

AUTHORS

The original version of UNIX awk was designed and implemented by Alfred Aho, Peter Weinberger, and Brian Kernighan of Bell Laboratories. Brian Kernighan continues to maintain and enhance it.

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 contributed a number of bug fixes.

David Trueman, with contributions from Arnold Robbins, made gawk com? patible with the new version of UNIX awk. Arnold Robbins is the cur? rent maintainer.

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

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

BUG REPORTS

If you find a bug in gawk, please send electronic mail to bug-gawk@gnu.org. Please include your operating system and its revi? sion, the version of gawk (from gawk --version), which C compiler you used to compile it, and a test program and data that are as small as possible for reproducing the problem.

Before sending a bug report, please do the following things. First, verify that you have the latest version of gawk. Many bugs (usually subtle ones) are fixed at each release, and if yours is out of date, the problem may already have been solved. Second, please see if set? ting the environment variable LC_ALL to LC_ALL=C causes things to be? have as you expect. If so, it's a locale issue, and may or may not re? ally be a bug. Finally, please read this man page and the reference manual carefully to be sure that what you think is a bug really is, in? stead of just a quirk in the language.

Whatever you do, do NOT post a bug report in comp.lang.awk. While the gawk developers occasionally read this newsgroup, posting bug reports there is an unreliable way to report bugs. Similarly, do NOT use a web forum (such as Stack Overflow) for reporting bugs. Instead, please use the electronic mail addresses given above. Really.

If you're using a GNU/Linux or BSD-based system, you may wish to submit a bug report to the vendor of your distribution. That's fine, but please send a copy to the official email address as well, since there's no guarantee that the bug report will be forwarded to the gawk main? tainer.

BUGS

The -F option is not necessary given the command line variable assign? ment feature; it remains only for backwards compatibility.

SEE ALSO

egrep(1), sed(1), getpid(2), getppid(2), getpgrp(2), getuid(2), ge? teuid(2), getgid(2), getegid(2), getgroups(2), printf(3), strftime(3), usleep(3)

The AWK Programming Language, Alfred V. Aho, Brian W. Kernighan, Peter

J. Weinberger, Addison-Wesley, 1988. ISBN 0-201-07981-X.

GAWK: Effective AWK Programming, Edition 5.1, 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") }'
```

ACKNOWLEDGEMENTS

Brian Kernighan provided valuable assistance during testing and debug? ging. We thank him.

COPYING PERMISSIONS

permission notice identical to this one.

Permission is granted to copy and distribute translations of this man?
ual page into another language, under the above conditions for modified
versions, except that this permission notice may be stated in a trans?
lation approved by the Foundation.