AWK REFERENCE

CONTENTS

This reference card was written by Arnold Robbins. Brian Kernighan and Michael Brennan reviewed it; we thank them for their help.

OTHER FSF PRODUCTS:

Free Software Foundation, Inc.
59 Temple Place — Suite 330
Boston, MA 02111-1307 USA
Phone: +1-617-542-5942
Fax (including Japan): +1-617-542-2652
E-mail: gnu@prep.ai.mit.edu

Free Software
Source Distributions on CD-ROM
Deluxe Distributions
Emacs, Gawk, Make and GDB Manuals
Emacs and GDB References

© Copyright, 1996, 1997 Free Software Foundation
59 Temple Place — Suite 330
Boston, MA 02111-1307 USA

DEFINITIONS

This card describes POSIX AWK, as well as the three freely available **awk** implementations (see FTP Information below). Common extensions (in two or more versions) are printed in light blue. Features specific to just one version—usually GNU AWK (**gawk**)—are printed in dark blue. Exceptions and deprecated features are printed in red. Features mandated by POSIX are printed in black.

Several type faces are used to clarify the meaning:
- **Courier Bold** is used for computer input.
- *Times Italic* is used to indicate user input and for syntactic placeholders, such as *variable* or *action*.
- Times Roman is used for explanatory text.

number – a floating point number as in ANSI C, such as **3**, **2.3**, **1.4e2** or **4.1E5**.

escape sequences – a special sequence of characters beginning with a backslash, used to describe otherwise unprintable characters. (See Escape Sequences below.)

string – a group of characters enclosed in double quotes. Strings may contain *escape sequences*.

regexp – a regular expression, either a regexp constant enclosed in forward slashes, or a dynamic regexp computed at run-time. Regexp constants may contain *escape sequences*.

name – a variable, array or function name.

entry(*N*) – entry *entry* in section *N* of the UNIX reference manual.

pattern – an expression describing an input record to be matched.

action – statements to execute when an input record is matched.

rule – a pattern-action pair, where the pattern or action may be missing.

COMMAND LINE ARGUMENTS (standard)

Command line arguments control setting the field separator, setting variables before the **BEGIN** rule is run, and the location of AWK program source code. Implementation-specific command line arguments change the behavior of the running interpreter.

-F *fs*	use *fs* for the input field separator.
-v *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.
-f *prog-file*	read the AWK program source from the file *prog-file*, instead of from the first command line argument. Multiple **-f** options may be used.
--	signal the end of options.

The following options are accepted by both Bell Labs **awk** and **gawk** (ignored by **gawk**, not in **mawk**).

-mf *val*	set the maximum number of fields to *val*
-mr *val*	set the maximum record size to *val*

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

COMMAND LINE ARGUMENTS (**gawk**)

The following options are specific to **gawk**. The **-W** forms are for full POSIX compliance.

--field-separator *fs*
just like **-F**

--assign *var***=***val* just like **-v**

--file *prog-file* just like **-f**

--traditional
--compat
-W compat
-W traditional turn off **gawk**-specific extensions (**--traditional** preferred).

--copyleft
--copyright
-W copyleft
-W copyright print the short version of the GNU copyright information on **stdout**.

--help
--usage
-W help
-W usage print a short summary of the available options on **stdout**, then exit zero.

--lint
-W lint warn about constructs that are dubious or non-portable to other **awk**s.

--lint-old
-W lint-old warn about constructs that are not portable to the original version of Unix **awk**.

--posix
-W posix disable common and GNU extensions. Enable *interval expressions* in regular expression matching (see Regular Expressions below).

--re-interval
-W re-interval enable *interval expressions* in regular expression matching (see Regular Expressions below). Useful if **--posix** is not specified.

--source '*text***'**
-W source '*text***'** use *text* as AWK program source code.

--version
-W version print version information on **stdout** and exit zero.

In compatibility mode, any other options are flagged as illegal, 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 **ARGV** for processing. This is most useful for running AWK programs via the **#!** executable interpreter mechanism.

COMMAND LINE ARGUMENTS (**mawk**)

The following options are specific to **mawk**.

-W dump print an assembly listing of the program to **stdout** and exit zero.

-W exec *file* read program text from *file*. No other options are processed. Useful with **#!**.

-W interactive unbuffer **stdout** and line buffer **stdin**. Lines are always records, ignoring **RS**

-W posix_space **\n** separates fields when **RS = ""**.

-W sprintf=*num* adjust the size of **mawk**'s internal **sprintf** buffer.

-W version print version and copyright on **stdout** and limit information on **stderr** and exit zero.

The options may be abbreviated using just the first letter, e.g., **-We**, **-Wv** and so on.

AWK PROGRAM EXECUTION

AWK programs are a sequence of pattern-action statements and optional function definitions.

pattern **{** *action statements* **}**
function *name***(***parameter list***)** **{** *statements* **}**

awk first reads the program source from the *prog-file*(s), if specified, from arguments to **--source**, or from the first non-option argument on the command line. The program text is read as if all the *prog-file*(s) and command line source texts had been concatenated.

AWK programs execute in the following order. First, all variable assignments specified via the **-v** option are performed. Next, **awk** executes the code in the **BEGIN** rules(s), if any, and then proceeds to read the files **1** through **ARGC - 1** in the **ARGV** array. (Adjusting **ARGC** and **ARGV** thus provides control over the input files that will be processed.) If there are no files named on the command line, **awk** reads the standard input.

If a command line argument 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 values to the variables **awk** uses to control how input is broken into fields and records. It is also useful for controlling state if multiple passes are needed over a single data file.

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

For each record in the input, **awk** tests to see if it matches any *pattern* in the AWK program. For each pattern that the record matches, the associated *action* is executed. The patterns are tested in the order they occur in the program.

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

If a program only has a **BEGIN** rule, no input files are processed. If a program only has an **END** rule, the input will be read.

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

LINES AND STATEMENTS

AWK is a line oriented language. The pattern comes first, and then the action. Action statements are enclosed in **{** and **}**. Either the pattern or the action may be missing, but not both. If the pattern is missing, the action will be executed for every input record. 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. Normally, a statement ends with a newline, but lines ending in a ",", **{**, **?**, **:**, **&&** or **||** are automatically continued. 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 will be 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 statements themselves.

REGULAR EXPRESSIONS

Regular expressions are the extended kind originally defined by **egrep**. Additional GNU regexp operators are supported by **gawk**. A *word-constituent* character is a letter, digit, or underscore (_).

Summary of Regular Expressions
In Decreasing Precedence

(*r***)**	regular expression (for grouping)
c	if non-special char, matches itself
*****c*	turn off special meaning of *c*
^	beginning of string (note: *not* line)
$	end of string (note: *not* line)
.	any single character, including newline
[...**]**	any one character in ... or range
[^...**]**	any one character not in ... or range
\y	word boundary
\B	middle of a word
\\<	beginning of a word
\\>	end of a word
\w	any word-constituent character
\W	any non-word-constituent character
\\`	beginning of a buffer (string)
\\'	end of a buffer (string)
*r******	zero or more occurrences of *r*
*r***+**	one or more occurrences of *r*
*r***?**	zero or one occurrences of *r*
*r***{***n***,***m***}**	*n* to *m* occurrences of *r* (POSIX: see note below)
r1 **|** *r2*	*r1* or *r2*

The *r***{***n***,***m***}** notation is called an *interval expression*. POSIX mandates it for AWK regexps, but most **awk**s don't implement it. Use **--re-interval** or **--posix** to enable this feature in **gawk**.

POSIX CHARACTER CLASSES (gawk)

In regular expressions, within character ranges (**[**...**]**), the notation **[[:***class***:]]** defines characters classes:

alnum	alphanumeric	**lower**	lower-case
alpha	alphabetic	**print**	printable
blank	space or tab	**punct**	punctuation
cntrl	control	**space**	whitespace
digit	decimal	**upper**	upper-case
graph	non-spaces	**xdigit**	hexadecimal

RECORDS

Normally, records are separated by newline characters. Assigning values to the built-in variable **RS** controls how records are separated. If **RS** is any single character, that character separates records. Otherwise, **RS** is a regular expression. (Not Bell Labs **awk**.) Text in the input that matches this regular expression will separate the record. **gawk** sets **RT** to the value of the input text that matched the regular expression. The value of **IGNORECASE** will also affect how records are separated when **RS** is a regular expression. If **RS** is set to the null string, then records are separated by one or more blank lines. When **RS** is set to the null string, the newline character always acts as a field separator, in addition to whatever value **FS** may have. **mawk** does not apply exceptional rules to **FS** when **RS = ""**.

FIELDS

As each input record is read, **awk** 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. Otherwise, **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. Leading and trailing whitespace are ignored. The value of **IGNORECASE** will also affect how fields are split when **FS** is a regular expression.

If the **FIELDWIDTHS** variable is set to a space separated list of numbers, each field is expected to have a fixed width, and **gawk** will split up the record using the specified widths. The value of **FS** is ignored. Assigning a new value to **FS** overrides the use of **FIELDWIDTHS**, and restores the default behavior.

Each field in the input record may be referenced by its position, **$1**, **$2** and so on. **$0** is the whole record. Fields may also be assigned new values.

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**) will increase the value of **NF**, create any intervening fields with the null string as their value, and cause the value of **$0** to be recomputed with the fields being separated by the value of **OFS**. References to negative numbered fields cause a fatal error. Decreasing the value of **NF** causes the trailing fields to be lost (not Bell Labs **awk**).

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

PATTERN ELEMENTS

AWK patterns may be one of the following.

BEGIN
END
expression
pat1 **,** *pat2*

BEGIN and **END** are special patterns that provide start-up and clean-up actions respectively. They must have actions. There can be multiple **BEGIN** and **END** rules; they are merged and executed as if there had just been one large rule. They may occur anywhere in a program, including different source files.

Expression patterns can be any expression, as described under Expressions.

The *pat1* **,** *pat2* pattern is called a *range pattern*. It matches all input records starting with a record that matches *pat1*, and continuing until a record that matches *pat2*, inclusive. It does not combine with any other pattern expression.

ACTION STATEMENTS

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*]
next
nextfile (not **mawk**)
{ *statements* **}**

ESCAPE SEQUENCES

Within strings constants (**"..."**) and regexp constants (**/.../**), escape sequences may be used to generate otherwise unprintable characters. This table lists the available escape sequences.

\a	alert (bell)	**\r**	carriage return
\b	backspace	**\t**	horizontal tab
\f	form feed	**\v**	vertical tab
\n	newline	****	backslash
*****ddd*	octal value *ddd*	**\x***hh*	hex value *hh*
\\"	double quote	**\/**	forward slash

VARIABLES

ARGC	number of command line arguments.
ARGIND	index in **ARGV** of current data file.
ARGV	array of command line arguments. Indexed from 0 to **ARGC** − 1. Dynamically changing the contents of **ARGV** can control the files used for data.
CONVFMT	conversion format for numbers, default value is **"%.6g"**.
ENVIRON	array containing the the current environment. The array is indexed by the environment variables, each element being the value of that variable.
ERRNO	contains a string describing the error when a redirection or read for **getline** fails, or if **close()** fails.
FIELDWIDTHS	white-space separated list of fieldwidths. Used to parse the input into fields of fixed width, instead of the value of **FS**.
FILENAME	name of the current input file. If no files given on the command line, **FILENAME** is "−". **FILENAME** is undefined inside the **BEGIN** rule (unless set by **getline**).
FNR	number of the input record in current input file.
FS	input field separator, a space by default (see Fields above).
IGNORECASE	if non-zero, all regular expression and string operations ignore case. In versions of **gawk** prior to 3.0, **IGNORECASE** only affected regular expression operations and **index()**.
NF	number of fields in the current input record.
NR	total number of input records seen so far.
OFMT	output format for numbers, **"%.6g"**, by default. Old versions of **awk** also used this for number to string conversion instead of **CONVFMT**.
OFS	output field separator, a space by default.
ORS	output record separator, a newline by default.
RS	input record separator, a newline by default (see Records above).
RT	record terminator. **gawk** sets **RT** to the input text that matched the character or regular expression specified by **RS**.
RSTART	index of the first character matched by **match()**; 0 if no match.
RLENGTH	length of the string matched by **match()**; −1 if no match.
SUBSEP	character(s) used to separate multiple subscripts in array elements, by default **"\034"**. (see Arrays below).

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

ARRAYS

An arrays subscript is an expression between square brackets (**[** and **]**). If the expression is a list (*expr*, *expr* ...), then the subscript is a string consisting of the concatenation of the (string) value of each expression, separated by the value of the **SUBSEP** variable. This simulates multi-dimensional arrays. For example:

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

assigns **"hello, world\n"** to the element of the array **x** indexed by the string **"A\034B\034C"**. All arrays in AWK are associative, i.e., indexed by string values.

Use the special operator **in** in an **if** or **while** statement to see if a particular value is an array index.

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

If the array has multiple subscripts, use **(i, j) in array**.

Use the **in** construct in a **for** loop to iterate over all the elements of an array.

Use the **delete** statement to delete an element from an array. Specifying just the array name without a subscript in the **delete** statement deletes the entire contents of an array.

EXPRESSIONS

Expressions are used as patterns, for controlling conditional action statements, and to produce parameter values when calling functions. Expressions may also be used as simple statements, particularly if they have side-effects such as assignment. Expressions mix *operands* and *operators*. Operands are constants, fields, variables, array elements, and the return values from function calls (both built-in and user-defined).

Regexp constants (**/***pat***/**), when used as simple expressions, i.e., not used on the right-hand side of **~** and **!~**, or as arguments to the **gensub()**, **gsub()**, **match()**, **split()**, and **sub()**, functions, mean **$0 ~ /***pat***/**.

The AWK operators, in order of decreasing precedence, are

(...)	grouping
$	field reference
++ --	increment and decrement, prefix and postfix
^ **	exponentiation
+ - !	unary plus, unary minus, and logical negation
*** / %**	multiplication, division, and modulus
+ -	addition and subtraction
space	string concatenation
< >	less than, greater than
<= >=	less than or equal, greater than or equal
!= ==	not equal, equal
~ !~	regular expression match, negated match
in	array membership
&&	logical AND, short circuit
||	logical OR, short circuit
?:	in-line conditional expression
= += -= *= /= %= ^= **=	assignment operators

CONVERSIONS AND COMPARISONS

Variables and fields may be (floating point) numbers, strings or both. Context determines how the value of a variable is interpreted. 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 0 to it; to force it to be treated as a string, concatenate it with the null string.

When a string must be converted to a number, the conversion is accomplished using *atof*(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.

Comparisons are performed 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. According to the POSIX standard, even if two strings are numeric strings, a numeric comparison is performed. However, this is clearly incorrect, and none of the three free **awk**s do this.

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()** that are numeric strings. The basic idea is that *user input*, and only user input, that looks numeric, should be treated that way.

Uninitialized variables have the numeric value 0 and the string value **""** (the null, or empty, string).

ISBN: 0-916151-97-2

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

INPUT CONTROL

close(*file***)**	close input file or pipe.
getline	set **$0** from next input record; set **NF**, **NR**, **FNR**.
getline < *file*	set **$0** from next record of *file*; set **NF**.
getline *v*	set *v* from next input record; set **NR**, **FNR**.
getline *v* **<** *file*	set *v* from next record of *file*.
cmd **|** **getline**	pipe into **getline**; set **$0**, **NF**.
cmd **|** **getline** *v*	pipe into **getline**; set *v*.

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

nextfile
stop processing the current input file. The next input record comes from the next input file. **FILENAME** and **ARGIND** are updated, **FNR** is reset to 1, and processing starts over with the first pattern in the AWK program. Upon end of input data, execute any **END** rule(s). Earlier versions of **gawk** used **next file**, as two words. This generates a warning message and will eventually be removed. **mawk** does not currently support **nextfile**.

getline returns 0 on end of file, and –1 on an error.

OUTPUT CONTROL

close(*file***)**
close output file or pipe.

fflush([*file*]**)**
flush any buffers associated with the open output file or pipe *file*. If *file* is missing, then standard output is flushed. If *file* is the null string, then all open output files and pipes are flushed (not Bell Labs **awk**).

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**.

printf *fmt***,** *expr-list*
format and print (see Printf Formats below).

system(*cmd***)**
execute the command *cmd*, and return the exit status (may not be available on non-POSIX systems).

I/O redirections may be used with both **print** and **printf**.

print "hello" > *file*
Print data to *file*. The first time the file is written to, it will be truncated. Subsequent commands append data.

print "hello" >> *file*
Append data to *file*. The previous contents of the file are not lost.

print "hello" | *cmd*
Print data down a pipeline to *cmd*.

PRINTF FORMATS

The **printf** statement and **sprintf()** function accept the following conversion specification formats:

%c	an ASCII character
%d	a decimal number (the integer part)
%i	a decimal number (the integer part)
%e	a floating point number of the form **[-]d.dddddde[+-]dd**
%E	like **%e**, but use **E** instead of **e**
%f	a floating point number of the form **[-]ddd.dddddd**
%g	use **%e** or **%f**, whichever is shorter, with nonsignificant zeros suppressed
%G	like **%g**, but use **%E** instead of **%e**
%o	an unsigned octal integer
%s	a character string
%x	an unsigned hexadecimal integer
%X	like **%x**, but use **ABCDEF** for 10–15
%%	A literal **%**; no argument is converted

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

-	left-justify the expression within its field.
space	for numeric conversions, prefix positive values with a space and negative values with a minus sign.
+	used before the *width* modifier means 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 some control letters.
%o	supply a leading zero.
%x, **%X**	supply a leading **0x** or **0X** for a nonzero result.
%e, **%E**, **%f**	the result always has a decimal point.
%g, **%G**	trailing zeros are not removed.
0	a leading zero acts as a flag, indicating output should be padded with zeroes instead of spaces. This applies even to non-numeric output formats. Only has an effect when the field width is wider than the value to be printed.
width	pad the field to this width. The field is normally padded with spaces. If the **0** flag has been used, pad with zeroes. The meaning of the *width* varies by control letter:
%d, **%o**, **%i**, **%u**, **%x**, **%X**	the minimum number of digits to print.
%e, **%E**, **%f**	the number of digits to print to the right of the decimal point.
%g, **%G**	the maximum number of significant digits.
%s	the maximum number of characters to print.

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

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

SPECIAL FILENAMES

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

"-"	standard input
/dev/stdin	standard input (not **mawk**)
/dev/stdout	standard output
/dev/stderr	standard error output

The following names are specific to **gawk**.

/dev/fd/*n*	file associated with the open file descriptor *n*

Other special filenames provide access to information about the running **gawk** process. Reading from these files returns a single record. The filenames and what they return are:

/dev/pid	process ID of current process
/dev/ppid	parent process ID of current process
/dev/pgrpid	process group ID of current process
/dev/user	a single newline-terminated record. The fields are separated with spaces. **$1** is the return value of *getuid*(2), **$2** is the return value of *geteuid*(2), **$3** is the return value of *getgid*(2) , and **$4** is the return value of *getegid*(2). Any additional fields are the group IDs returned by *getgroups*(2). Multiple groups may not be supported on all systems.

These filenames will become obsolete in **gawk** 3.1. Be aware that you will have to change your programs.

NUMERIC FUNCTIONS

atan2(*y***,** *x***)**	the arctangent of *y/x* in radians.
cos(*expr***)**	the cosine of *expr*, which is in radians.
exp(*expr***)**	the exponential function (*e* ^ *x*).
int(*expr***)**	truncates to integer.
log(*expr***)**	the natural logarithm function (base *e*).
rand()	a random number between 0 and 1.
sin(*expr***)**	the sine of *expr*, which is in radians.
sqrt(*expr***)**	the square root function.
srand([*expr*]**)**	uses *expr* as a new seed for the random number generator. If no *expr*, the time of day is used. Returns previous seed for the random number generator.

STRING FUNCTIONS

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**, replace all matches of *r* with *s*. Otherwise, *h* is a number indicating which match of *r* to replace. If no *t* is supplied, **$0** is used 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 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 replaced with the text that was actually matched. Use **\&** to get a literal **&**. See *The GNU Awk User's Guide* for a fuller discussion of the rules for **&**'s and backslashes in the replacement text of **gensub()**, **sub()** and **gsub()**

index(*s***,** *t***)**
returns the index of the string *t* in the string *s*, or 0 if *t* is not present.

length([*s*]**)**
returns the length of the string *s*, or the length of **$0** if *s* is not supplied.

match(*s***,** *r***)**
returns the position in *s* where the regular expression *r* occurs, or 0 if *r* is not present, and sets the values of variables **RSTART** and **RLENGTH**.

split(*s***,** *a* [**,** *r*]**)**
splits the string *s* into the array *a* using the regular expression *r*, and returns the number of fields. If *r* is omitted, **FS** is used instead. The array *a* is cleared first. Splitting behaves identically to field splitting. (See Fields, above.)

sprintf(*fmt***,** *expr-list***)**
prints *expr-list* according to *fmt*, and returns the resulting string.

sub(*r***,** *s* [**,** *t*]**)**
just like **gsub()**, but only the first matching substring is replaced.

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

tolower(*str***)**
returns a copy of the string *str*, with all the upper-case characters in *str* translated to their corresponding lower-case counterparts. Non-alphabetic characters are left unchanged.

toupper(*str***)**
returns a copy of the string *str*, with all the lower-case characters in *str* translated to their corresponding upper-case counterparts. Non-alphabetic characters are left unchanged.

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

TIME FUNCTIONS (gawk)

gawk provides the following functions for obtaining time stamps and formatting them.

strftime([*format* [**,** *timestamp*]]**)**
: formats *timestamp* according to the specification in *format*. 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) will be used.

systime()
: returns the current time of day as the number of seconds since the Epoch.

USER-DEFINED FUNCTIONS

Functions in AWK are defined as follows:

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

Functions are executed when they are called from within expressions in either patterns or actions. Actual parameters supplied in the function call 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:

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

The left parenthesis in a function call is required to immediately follow the function name without any intervening white space. This is to avoid a syntactic ambiguity with the concatenation operator. This restriction does not apply to the built-in functions.

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.

The word **func** may be used in place of **function**. Note: This usage is deprecated.

BUG REPORTS

If you find a bug in this reference card, please report it via electronic mail to **arnold@gnu.ai.mit.edu**.

ENVIRONMENT VARIABLES (gawk)

The environment variable **AWKPATH** specifies a search path to use when finding source files named with the **-f** option. The default path is **".:/usr/local/share/awk"**, if this variable does not exist. (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.

If **POSIXLY_CORRECT** exists in the environment, then **gawk** behaves exactly as if **--posix** had been specified on the command line.

HISTORICAL FEATURES (gawk)

gawk supports two features of historical AWK implementations. First, it is possible to call the **length()** built-in function not only with no argument, but even without parentheses. This feature is marked as "deprecated" in the POSIX standard, and **gawk** will issue a warning about its use if **--lint** is specified on the command line.

The other feature is the use of **continue** or **break** statements outside the body of a **while**, **for**, or **do** loop. Historical AWK implementations have treated such usage as equivalent to the **next** statement. **gawk** will support this usage if **--traditional** has been specified.

FTP INFORMATION

Host: **ftp.gnu.ai.mit.edu**
File: **/pub/gnu/gawk-3.0.3.tar.gz**
GNU **awk** (**gawk**). There may be a later version.

Host: **netlib.bell-labs.com**
File: **/netlib/research/awk.bundle.Z**
Bell Labs **awk**. This version requires an ANSI C compiler; GCC (the GNU C compiler) works well.

Host: **ftp.whidbey.net**
File: **/pub/brennan/mawk1.3.3.tar.gz**
Michael Brennan's **mawk**. There may be a newer version.

COPYING PERMISSIONS

Copyright © 1996, 1997 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this reference card provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this reference card under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this reference card into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

Copyright 05-13-97 22:22:06, FSF, Inc. (all)

Copyright 05-13-97 22:22:06, FSF, Inc. (all)