AWK (Aho, Kernighan, and Weinberger) Summary

Predefined Variable Summary:

		Support:		
Variable	**Description**	**AWK**	**NAWK**	**GAWK**
FS	Input **F**ield **S**eparator, a space by default.	√	√	√
OFS	**O**utput **F**ield **S**eparator, a space by default.	√	√	√
NF	The **N**umber of **F**ields in the current input record.	√	√	√
NR	The total **N**umber of input **R**ecords seen so far.	√	√	√
RS	**R**ecord **S**eparator, a newline by default.	√	√	√
ORS	**O**utput **R**ecord **S**eparator, a newline by default.	√	√	√
FILENAME	The name of the current input file. If no files are specified on the command line, the value of **FILENAME** is "-". However, **FILENAME** is undefined inside the **BEGIN** block (unless set by ***getline***).	√	√	√
ARGC	The number of command line arguments (does not include options to gawk, or the program source). Dynamically changing the contents of **ARGV** control the files used for data.	–	√	√
ARGV	Array of command line arguments. The array is indexed from 0 to **ARGC** - 1.	–	√	√
ARGIND	The index in **ARGV** of the current file being processed.	–	–	√
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 **CONVFMT** variable is used to specify the format when converting a number to a string. Default: "**%.6g**"	–	–	√
ENVIRON	An array containing the values of the current environment.	–	–	√
ERRNO	If a system error occurs either doing a redirection for **getline**, during a read for ***getline***, or during a **close()**, then **ERRNO** will contain a string describing the error. The value is subject to translation in non-English locales.	–	–	√
FIELDWIDTHS	A white-space separated list of fieldwidths. When set, *gawk* parses the input into fields of fixed width, instead of using the value of the **FS** variable as the field separator.	–	–	√
FNR	Contains number of lines read, but is reset for each file read.	–	√	√
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**, record separating with RS, regular expression matching with **~** and **!~**, and the **gensub()**, **gsub()**, **index()**, **match()**, **split()**, and **sub()** built-in functions all ignore case when doing regular expression operations. NOTE: Array subscripting is not affected. However, the **asort()** and **asorti()** functions are affected.	–	–	√
LINT	Provides dynamic control of the **--lint** option from within an AWK program. When true, *gawk* prints lint warnings.	–	–	√
OFMT	The default output format for numbers. Default: "**%.6g**"	–	√	√
PROCINFO	The elements of this array provide access to information about the running AWK program. PROCINFO["egid"] the value of the getegid(2) system call. PROCINFO["euid"] the value of the geteuid(2) system call. PROCINFO["FS"] "**FS**" if field splitting with FS is in effect, or "**FIELDWIDTHS**" if field splitting with **FIELDWIDTHS** is in effect. PROCINFO["gid"] the value of the getgid(2) system call. PROCINFO["pgrpid"] the process group ID of the current process. PROCINFO["pid"] the process ID of the current process. PROCINFO["ppid"] the parent process ID of the current process. PROCINFO["uid"] the value of the getuid(2) system call.	–	–	√
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.	–	√	√
RLENGTH	The length of the string matched by **match();** -1 if no match.	–	√	√
SUBSEP	The character used to separate multiple subscripts in array elements. Default: "**\034**" (non-printable character, dec: 28, hex: 1C)	–	√	√
TEXTDOMAIN	The text domain of the AWK program; used to find the localized translations for the program's strings.	–	–	√

peter@catonmat.net
http://www.catonmat.net
good coders code, great reuse

Variable is supported: √
Variable is not supported: –

Command line argument summary:

Argument	Description
-F *fs* **--field-sepearator *fs***	Use ***fs*** for the input field separator (the value of the **FS** predefined variable).
-v *var=val* **--assign *var=val***	Assign the value ***val*** to the variable ***var***, before execution of the program begins. Such variable values are available to the **BEGIN** block of an *AWK* program.
-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.
-mf *NNN* **-mr *NNN***	Set various memory limits to the value **NNN**. The **f** flag sets the maximum number of fields, and the r flag sets the maximum record size. (Ignored by *gawk*, since *gawk* has no pre-defined limits)
-W compat **-W traditional** **--compat** **--traditional**	Run in compatibility mode. In compatibility mode, *gawk* behaves identically to UNIX *awk*; none of the GNU-specific extensions are recognized.
-W copyleft **-W copyright** **--copyleft** **--copyright**	Print the short version of the GNU copyright information message on the standard output and exit successfully.
-W dump-variables[=*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.
-W help **-W usage** **--help** **--usage**	Print a relatively short summary of the available options on the standard output.
-W lint[=*value*] **--lint[=*value*]**	Provide warnings about constructs that are dubious or non-portable to other AWK impl's. With argument **fatal**, lint warnings become fatal errors. With an optional argument of **invalid**, only warnings about things that are actually invalid are issued. (This is not fully implemented yet.)
-W lint-old **--lint-old**	Provide warnings about constructs that are not portable to the original version of Unix *awk*.
-W gen-po **--gen-po**	Scan and parse the AWK program, and generate a GNU .po format file on standard output with entries for all localizable strings in the program. The program itself is not executed.
-W non-decimal-data **--non-decimal-data**	Recognize octal and hexadecimal values in input data.
-W posix **--posix**	This turns on compatibility mode, with the following additional restrictions: • \x escape sequences are not recognized. • Only space and tab act as field separators when **FS** is set to a single space, new-line does not. • 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 ^=. • The ***fflush()*** function is not available.
-W profile[=*prof_file*] **--profile[=*prof_file*]**	Send profiling data to ***prof_file***. The default is awkprof.out. When run with *gawk*, the profile is just a "pretty printed" version of the program. When run with *pgawk*, the profile contains execution counts of each statement in the program in the left margin and function call counts for each user-defined function.
-W re-interval **--re-interval**	Enable the use of interval expressions in regular expression matching. Interval expressions were not traditionally available in the AWK language.
-W source *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 **--file** options) with source code entered on the command line.
-W version **--version**	Print version information for this particular copy of *gawk* on the standard output.
--	Signal the end of options. This is useful to allow further arguments to the AWK program itself to start with a "-". This is mainly for consistency with the argument parsing convention used by most other POSIX programs.

Statements and Functions:

I/O Statements	Description
close(*file* [, *how*])	Close ***file***, ***pipe*** or ***co-process***. The optional ***how*** should only be used when closing one end of a two-way pipe to a co-process. It must be a string value, either "to" or "from".
getline	Set **$0** from next input record; set **NF**, **NR**, **FNR**. Returns 0 on EOF and –1 on an error. Upon an error, ERRNO contains a string describing the problem.
getline <*file*	Set **$0** from next record of ***file***; set **NF**.
getline *var*	Set ***var*** from next input record; set **NR**, **FNR**.
getline *var* <*file*	Set ***var*** from next record of ***file***.
command | getline [*var*]	Run **command** piping the output either into **$0** or ***var***, as above. If using a pipe or co-process to **getline**, or from print or **printf** within a loop, you must use **close()** to create new instances
command |& getline [*var*]	Run command as a co-process piping the output either into **$0** or ***var***, as above. Co-processes are a *gawk* extension.
next	Stop processing the current input record. The next input record is read and processing starts over with the first pattern in the AWK program. If the end of the input data is reached, the **END** block(s), if any, are executed.
nextfile	Stop processing the current input file. The next input record read 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. If the end of the input data is reached, the **END** block(s), are executed.
print	Prints the current record. The output record is terminated with the value of the **ORS** variable.
print *expr-list*	Prints expressions. Each expression is separated by the value of the **OFS** variable. The output record is terminated with the value of the **ORS** variable.
print *expr-list* >*file*	Prints expressions on file. Each expression is separated by the value of the OFS variable. The output record is terminated with the value of the ORS variable.
printf *fmt*, *expr-list*	Format and print.
printf *fmt*, *expr-list* >*file*	Format and print on file.
system(*cmd-line*)	Execute the command ***cmd-line***, and return the exit status.
fflush([*file*])	Flush any buffers associated with the open output file or pipe file. If file is missing, then stdout is flushed. If file is the null string, then all open output files and pipes have their buffers flushed.
print ... >> file	appends output to the file.
print ... | command	writes on a pipe.
print ... |& command	sends data to a co-process.

Numeric Functions	Description
atan2(*y, x*)	Returns the arctangent of ***y***/***x*** in radians.
cos(*expr*)	Returns the cosine of ***expr***, which is in radians.
exp(*expr*)	The exponential function.
int(*expr*)	Truncates to integer.
log(*expr*)	The natural logarithm function.
rand()	Returns a random number N, between 0 and 1, such that 0 <= N < 1.
sin(*expr*)	Returns 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*** is provided, the time of day is used. The return value is the previous seed for the random number generator.

Bit Manipulations Functions	Description
and(*v1*, *v2*)	Return the bitwise AND of the values provided by ***v1*** and ***v2***.
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 by ***v1*** and ***v2***.
rshift(*val*, *count*)	Return the value of ***val***, shifted right by ***count*** bits.
xor(*v1*, *v2*)	Return the bitwise XOR of the values provided by ***v1*** and ***v2***.

I18N functions
bindtextdomain(*directory* [, *domain*])
Specifies the directory where gawk looks for the .mo files. 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*]])
Returns the translation of string in text domain ***domain*** for locale category ***category***. The default value for ***domain*** is the current value of TEXTDOMAIN. The default value for ***category*** is "LC_MESSAGES". If you supply a value for ***category***, it must be a string equal to one of the known locale categories. You must also supply a text domain. Use TEXTDOMAIN if you want to use the current domain.
dcngettext(*string1* , *string2* , *number* [, *domain* [, *category*]])
Returns 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. You must also supply a text domain. Use TEXTDOMAIN if you want to use the current domain.

String Functions	Description
asort(*s* [, *d*])	Returns the number of elements in the source array ***s***. The contents of ***s*** are sorted using *gawk's* normal rules for comparing values, and the indexes of the sorted values of s are replaced with sequential integers starting with 1. If the optional destination array ***d*** is specified, then ***s*** is first duplicated into ***d***, and then ***d*** is sorted, leaving the indexes of the source array ***s*** unchanged.
asorti(*s* [, *d*])	Returns the number of elements in the source array ***s***. The behavior is the same as that of **asort()**, except that the array indices are used for sorting, not the array values. When done, the array is indexed numerically, and the values are those of the original indices. The original values are lost; thus provide a second array if you wish to preserve the original.
gensub(*r*, *s*, *h* [, *t*])	Search the target string ***t*** for matches of the regular expression ***r***. If ***h*** is a string beginning with *g* or *G*, then replace all matches of ***r*** with ***s***. Otherwise, ***h*** is a number indicating which match of ***r*** to replace. If ***t*** is not supplied, **$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 &. (This must be typed as "\\&")
index(*s*, *t*)	Returns the index of the string ***t*** in the string ***s***, or 0 if t is not present. (This implies that character indices start at one.)
length([*s*])	Returns the length of the string **s**, or the length of **$0** if s is not supplied.
match(*s*, *r* [, *a*])	Returns the position in ***s*** where the regular expression ***r*** occurs, or 0 if ***r*** is not present, and sets 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 0'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 matching substring.
split(*s*, *a* [, *r*])	Splits the string ***s*** into the array ***a*** on 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.
sprintf(*fmt*, *expr-list*)	Prints ***expr-list*** according to ***fmt***, and returns the resulting string.
strtonum(*str*)	Examines ***str***, and returns its numeric value. If ***str*** begins with a leading 0, **strtonum()** assumes that ***str*** is an octal number. If ***str*** begins with a leading 0x or 0X, **strtonum()** assumes that ***str*** is a hexadecimal number.
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.

Time Functions	Description
mktime(*datespec*)	Turns ***datespec*** into a time stamp of the same form as returned by **systime()**. 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 respectively 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, and 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 preceding year 1 and year -1 preceding year 0. The time is assumed to be in the local timezone. If the 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 negative (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*]])	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) is used. See the specification for the strftime() function in ANSI C for the format conversions that are guaranteed to be available. A public-domain version of strftime(3) and a man page for it come with gawk; if that version was used to build gawk, then all of the conversions described in that man page are available to gawk.
systime()	Returns the current time of day as the number of seconds since the Epoch (1970-01-01 00:00:00 UTC on POSIX systems).