NAME

awk – pattern scanning and processing language

SYNOPSIS

/usr/bin/awk [**–f** *progfile*] [**–F***c*] [**'***prog***'**] [*parameters*] [*filename*. . .]

/usr/xpg4/bin/awk [**–F** *ERE*] [**–v** *assignment* . . .] *'program'* | **–f** *progfile* . . .
[*argument* . . .]

AVAILABILITY

SUNWesu

DESCRIPTION

/usr/xpg4/bin/awk is described on the **nawk**(1) manual page.

/usr/bin/awk scans each input *filename* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (′) to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *filename* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the **–f** *progfile* option. Input files are read in order; if there are no files, the standard input is read. The file name **'–'** means the standard input.

OPTIONS

–f *progfile* **awk** uses the set of patterns it reads from *progfile*.

–F*c* Use the character *c* as the field separator (FS) character. See the discussion of **FS** below.

USAGE

Input Lines

Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. Any *filename* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. *Variables* assigned in this manner are not available inside a **BEGIN** rule, and are assigned after previously specified files have been read.

An input line is normally made up of fields separated by white spaces. (This default can be changed by using the **FS** built-in variable or the **–F***c* option.) The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if **FS** is assigned a value that does not include any of the white spaces, then leading blanks are not ignored. The fields are denoted **$1**, **$2**, **. . .** ; **$0** refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form:

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

Either pattern or action may be omitted. If there is no action, the matching line is printed. If there is no pattern, the action is performed on every input line. Pattern-action statements are separated by newlines or semicolons.

Patterns are arbitrary Boolean combinations (**!**, | | , **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular_expression

where a *relop* is any of the six relational operators in C, and a *matchop* is either ˜ (contains) or **!˜** (does not contain). An *expression* is an arithmetic expression, a relational expression, the special expression

var **in** array

or a Boolean combination of these.

Regular expressions are as in **egrep**(1). In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between the occurrence of the first pattern to the occurrence of the second pattern.

The special patterns **BEGIN** and **END** may be used to capture control before the first input line has been read and after the last input line has been read respectively. These keywords do not combine with any other patterns.

Built-in Variables

Built-in variables include:

FILENAME	name of the current input file
FS	input field separator regular expression (default blank and tab)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default **%.6g**)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

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

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

Statements are terminated by semicolons, newlines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators **+**, **–**, **∗**, **/**, **%**, **^** and concatenation (indicated by a blank). The operators **++ –– += –= ∗= /= %= ^= > >= < <= == != ?:** are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (**""**), with the usual C escapes recognized within.

The **print** statement prints its arguments on the standard output, or on a file if **>***expression* is present, or on a pipe if '|*cmd*' is present. The output resulted from the print statement is terminated by the output record separator with each argument separated by the current output field separator. The **printf** statement formats its expression list according to the format (see **printf**(3S)).

The mathematical functions: **exp**, **log**, **sqrt**, are built-in.

Other built-in functions include:

index(*s***,** *t***)** returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

int(*s***)** truncates *s* to an integer value. If *s* is not specified, $0 is used.

length(*s***)** returns the length of its argument taken as a string, or of the whole line if there is no argument.

match(*s***,** *re***)** returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all.

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

sprintf(*fmt*, *expr*, *expr*, . . . **)**
formats the expressions according to the **printf**(3S) format given by *fmt* and returns the resulting string.

substr(*s*, *m*, *n***)** returns the *n*-character substring of *s* that begins at position *m*.

The input/output built-in function is:

getline sets **$0** to the next input record from the current input file. **getline** returns 1 for successful input, 0 for end of file, and –1 for an error.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

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

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

Add up first column, print sum and average:

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

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

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

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print a file, filling in page numbers starting at 5:

```
/Page/  { $2 = n++; }
        { print }
```

Assuming this program is in a file named **prog**, the following command line prints the file **input** numbering its pages starting at 5: **awk −f prog n=5 input**.

ENVIRONMENT

If any of the **LC_*** variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **awk** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_*** variables. If none of the above variables is set in the environment, the "C" (U.S style) locale determines how **awk** behaves.

LC_CTYPE
: determines how **awk** handles characters. When **LC_CTYPE** is set to a valid value, **awk** can display and handle text and filenames containing valid characters for that locale. **awk** can display and handle Extended Unix Code (EUC) characters where any character can be 1, 2, or 3 bytes wide. **awk** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

LC_MESSAGES
: determines how diagnostic and informative messages are presented. This includes the language and style of the messages, and the correct form of affirmative and negative responses. In the "C" locale, the messages are presented in the default form found in the program itself (in most cases, U.S/English).

SEE ALSO

egrep(1), **grep**(1), **nawk**(1), **sed**(1), **printf**(3S), **environ**(5)

NOTES

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.