NAME

formats – file format notation

DESCRIPTION

Utility descriptions use a syntax to describe the data organization within files—stdin, stdout, stderr, input files, and output files—when that organization is not otherwise obvious. The syntax is similar to that used by the **printf**(3S) function. When used for stdin or input file descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the **scanf**(3S) function to read the input file.

Format

The description of an individual record is as follows:

"<*format*>", [<*arg1*>, <*arg2*>, . . ., <*argn*>]

The *format* is a character string that contains three types of objects defined below:

characters Characters that are not *escape sequences* or *conversion specifications*, as described below, are copied to the output.

escape sequences Represent non-graphic characters.

conversion specifications
Specifies the output format of each argument. (See below.)

The following characters have the following special meaning in the format string:

" " (An empty character position.) One or more blank characters

Δ Exactly one space character.

The notation for spaces allows some flexibility for application output. Note that an empty character position in *format* represents one or more blank characters on the output (not *white space*, which can include newline characters). Therefore, another utility that reads that output as its input must be prepared to parse the data using **scanf**(3S), **awk**(1), and so forth. The Δ character is used when exactly one space character is output.

Escape Sequences

The following table lists escape sequences and associated actions on display devices capable of the action.

Escape Sequence	Represents Character	Terminal Action
****	backslash	None.
\a	alert	Attempts to alert the user through audible or visible notification.
\b	backspace	Moves the printing position to one column before the current position, unless the current position is the start of a line.
\f	form-feed	Moves the printing position to the initial printing position of the next logical page.
\n	newline	Moves the printing position to the start of the next line.
\r	carriage-return	Moves the printing position to the start of the current line.

\t	tab	Moves the printing position to the next tab position on the current line. If there are no more tab positions left on the line, the behaviour is undefined.
\v	vertical-tab	Moves the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behaviour is undefined.

Conversion Specifications

Each conversion specification is introduced by the percent-sign character (%). After the character %, the following appear in sequence:

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

field width An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it is padded on the left (or right, if the left-adjustment flag (−), described below, has been given to the field width).

precision Gives the minimum number of digits to appear for the d, o, i, u, x or X conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the e and f conversions, the maximum number of significant digits for the g conversion; or the maximum number of bytes to be written from a string in s conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero.

conversion characters
A conversion character (see below) that indicates the type of conversion to be applied.

flags

The *flags* and their meanings are:

− The result of the conversion is left-justified within the field.

\+ The result of a signed conversion always begins with a sign (+ or −).

<*space*> If the first character of a signed conversion is not a sign, a space character is prefixed to the result. This means that if the space character and + flags both appear, the space character flag is ignored.

The value is to be converted to an alternative form. For c, d, i, u, and s conversions, the behaviour is undefined. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x or X conversion, a non-zero result has 0x or 0X prefixed to it, respectively. For e, E, f, g, and G conversions, the result always contains a radix character, even if no digits follow the radix character. For g and G conversions, trailing zeros are not removed from the result as they usually are.

0 For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and − flags both appear, the 0 flag is ignored. For d, i, o, u, x and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behaviour is undefined.

Conversion Characters

Each conversion character results in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are ignored.

The *conversion characters* and their meanings are:

d,i,o,u,x,X The integer argument is written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers convert to signed decimal in the style **[−]***dddd*. The x conversion uses the numbers and letters 0123456789abcdef and the X conversion uses the numbers and letters 0123456789ABCDEF. The *precision* component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters. If both the field width and precision are omitted, the implementation may precede, follow or precede and follow numeric arguments of types d, i and u with blank characters; arguments of type o (octal) may be preceded with leading zeros.

The treatment of integers and spaces is different from the **printf**(3S) function in that they can be surrounded with blank characters. This was done so that, given a format such as:

"%d\n",<*foo*>

the implementation could use a **printf( )** call such as:

printf("%6d\n", *foo*);

and still conform. This notation is thus somewhat like **scanf( )** in addition to **printf( ).**

f The floating point number argument is written in decimal notation in the style **[−]***ddd.ddd*, where the number of digits after the radix character (shown here as a decimal point) is equal to the *precision* specification. The **LC_NUMERIC** locale category determines the radix character to use in this format. If the *precision* is omitted from the argument, six digits are written after the radix character; if the *precision* is explicitly 0, no radix character appears.

e,E The floating point number argument is written in the style **[−]***d.ddd*e±*dd* (the symbol ± indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The **LC_NUMERIC** locale category determines the radix character to use in this format. When the precision is missing, six digits are written after the radix character; if the precision is 0, no radix character appears. The E conversion character produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits are written as necessary.

g,G The floating point number argument is written in style f or e (or in style E in the case of a G conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted: style g is used only if the exponent resulting from the conversion is less than –4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character appears only if it is followed by a digit.

c The integer argument is converted to an **unsigned char** and the resulting byte is written.

s The argument is taken to be a string and bytes from the string are written until the end of the string or the number of bytes indicated by the *precision* specification of the argument is reached. If the precision is omitted from the argument, it is taken to be infinite, so all bytes up to the end of the string are written.

% Write a % character; no argument is converted.

In no case does a non-existent or insufficient *field width* cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term *field width* should not be confused with the term *precision* used in the description of %s.

One difference from the C function **printf()** is that the l and h conversion characters are not used. There is no differentiation between decimal values for type **int**, type **long**, or type **short**. The specifications %d or %i should be interpreted as an arbitrary length sequence of digits. Also, no distinction is made between single precision and double precision numbers (**float** or **double** in C). These are simply referred to as floating point numbers.

Many of the output descriptions use the term *line*, such as:

"%s", <*input line*>

Since the definition of *line* includes the trailing newline character already, there is no need to include a **\n** in the format; a double newline character would otherwise result.

EXAMPLES

To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where <*weekday*> and <*month*> are strings:

"%s,Δ%sΔ%d,Δ%d:%.2d\n",<*weekday*>,<*month*>,<*day*>,<*hour*>,<*min*>

To show π written to 5 decimal places:

"piΔ=Δ%.5f\n",<*value of* π>

To show an input file format consisting of five colon-separated fields:

"%s:%s:%s:%s:%s\n",<*arg1*>,<*arg2*>,<*arg3*>,<*arg4*>,<*arg5*>

SEE ALSO

awk(1), **printf**(1), **printf**(3S), **scanf**(3S)