NAME

printf - formatted output

SYNOPSIS

printf *format* [*arguments* ...]

DESCRIPTION

The **printf** utility formats and prints its arguments, after the first, under control of the *format*. The *format* is a character string which contains three types of objects: plain characters, which are simply copied to standard output, character escape sequences which are converted and copied to the standard output, and format specifications, each of which causes printing of the next successive *argument*.

The *arguments* after the first are treated as strings if the corresponding format is either **c**, **b** or **s**; otherwise it is evaluated as a C constant, with the following extensions:

- A leading plus or minus sign is allowed.
- If the leading character is a single or double quote, the value is the character code of the next character.

The format string is reused as often as necessary to satisfy the *arguments*. Any extra format specifications are evaluated with zero or the null string.

Character escape sequences are in backslash notation as defined in the ANSI X3.159-1989 ("ANSI C89"), with extensions. The characters and their meanings are as follows:

\a	Write a <bell> character.
\b	Write a <backspace> character.
\f	Write a <form-feed> character.
\n	Write a <new-line> character.
\r	Write a <carriage return> character.
\t	Write a <tab> character.
\v	Write a <vertical tab> character.
\’	Write a <single quote> character.
\\	Write a backslash character.
num	Write a byte whose value is the 1-, 2-, or 3-digit octal number *num*. Multibyte characters can be constructed using multiple *num* sequences.

Each format specification is introduced by the percent character (‘‘%’’). The remainder of the format specification includes, in the following order:

Zero or more of the following flags:

A ‘#’ character specifying that the value should be printed in an ‘‘alternate form’’. For **b**, **c**, **d**, **s** and **u** formats, this option has no effect. For the **o** formats the precision of the number is increased to force the first character of the output string to a zero. For the **x** (**X**) format, a non-zero result has the string 0x (0X) prepended to it. For **a**, **A**, **e**, **E**, **f**, **F**, **g** and **G** formats, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point only appears in the results of those formats if a digit follows the decimal point). For **g** and **G** formats, trailing zeros are not removed from the result as they would otherwise be;

- A minus sign ‘-’ which specifies *left adjustment* of the output in the indicated field;

+ A ‘+’ character specifying that there should always be a sign placed before the number when using signed formats.

‘ ’ A space specifying that a blank should be left before a positive number for a signed format. A ‘+’ overrides a space if both are used;

0 A zero ‘0’ character indicating that zero-padding should be used rather than blank-padding. A ‘-’ overrides a ‘0’ if both are used;

Field Width:

An optional digit string specifying a *field width*; if the output string has fewer bytes than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width (note that a leading zero is a flag, but an embedded zero is part of a field width);

Precision:

An optional period, ‘**.**’, followed by an optional digit string giving a *precision* which specifies the number of digits to appear after the decimal point, for **e** and **f** formats, or the maximum number of bytes to be printed from a string; if the digit string is missing, the precision is treated as zero;

Format:

A character which indicates the type of format to use (one of **diouxXfFeEgGaAcsb**). The uppercase formats differ from their lowercase counterparts only in that the output of the former is entirely in uppercase. The floating-point format specifiers (**fFeEgGaA**) may be prefixed by an **L** to request that additional precision be used, if available.

A field width or precision may be '*****' instead of a digit string. In this case an *argument* supplies the field width or precision.

The format characters and their meanings are:

diouXx The *argument* is printed as a signed decimal (d or i), unsigned octal, unsigned decimal, or unsigned hexadecimal (X or x), respectively.

fF The *argument* is printed in the style '[-]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed. The values infinity and NaN are printed as 'inf' and 'nan', respectively.

eE The *argument* is printed in the style **e** '[*-d.ddd+-dd*]' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. The values infinity and NaN are printed as 'inf' and 'nan', respectively.

gG The *argument* is printed in style **f** (**F**) or in style **e** (**E**) whichever gives full precision in minimum space.

aA The *argument* is printed in style '[*-h.hhh+-pd*]' where there is one digit before the hexadecimal point and the number after is equal to the precision specification for the argument; when the precision is missing, enough digits are produced to convey the argument's exact double-precision floating-point representation. The values infinity and NaN are printed as 'inf' and 'nan', respectively.

c The first byte of *argument* is printed.

s Bytes from the string *argument* are printed until the end is reached or until the number of bytes indicated by the precision specification is reached; however if the precision is 0 or missing, the string is printed entirely.

b As for **s**, but interpret character escapes in backslash notation in the string *argument*. The permitted escape sequences are slightly different in that octal escapes are **\0***num* instead of *num* and that an additional escape sequence **\c** stops further output from this **printf** invocation.

n$ Allows reordering of the output according to *argument*.

% Print a ‘%’; no argument is used.

The decimal point character is defined in the program’s locale (category LC_NUMERIC).

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width.

Some shells may provide a builtin **printf** command which is similar or identical to this utility. Consult the builtin(1) manual page.

EXIT STATUS

The **printf** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Print the string "hello":

```
$ printf "%s\n" hello
hello
```

Same as above, but notice that the format string is not quoted and hence we do not get the expected behavior:

```
$ printf %s\n hello
hellon$
```

Print arguments forcing sign only for the first argument:

```
$ printf "%+d\n%d\n%d\n" 1 -2 13
+1
-2
13
```

Same as above, but the single format string will be applied to the three arguments:

```
$ printf "%+d\n" 1 -2 13
+1
-2
+13
```

Print number using only two digits after the decimal point:

```
$ printf "%.2f\n" 31.7456
31.75
```

COMPATIBILITY

The traditional BSD behavior of converting arguments of numeric formats not beginning with a digit to the ASCII code of the first character is not supported.

SEE ALSO

builtin(1), echo(1), sh(1), printf(3)

STANDARDS

The **printf** command is expected to be compatible with the IEEE Std 1003.2 ("POSIX.2") specification.

HISTORY

The **printf** command appeared in 4.3BSD-Reno. It is modeled after the standard library function, printf(3).

CAVEATS

ANSI hexadecimal character constants were deliberately not provided.

Trying to print a dash ("-") as the first character causes **printf** to interpret the dash as a program argument. **--** must be used before *format*.

If the locale contains multibyte characters (such as UTF-8), the **c** format and **b** and **s** formats with a precision may not operate as expected.

BUGS

Since the floating point numbers are translated from ASCII to floating-point and then back again, floating-point precision may be lost. (By default, the number is translated to an IEEE-754 double-precision value before being printed. The **L** modifier may produce additional precision, depending on the hardware platform.)

The escape sequence \000 is the string terminator. When present in the argument for the **b** format, the argument will be truncated at the \000 character.

Multibyte characters are not recognized in format strings (this is only a problem if ‘%’ can appear inside a multibyte character).