printf Cheatsheet

By Dejan Panovski • Updated on Nov 23, 2026

printf reference for Bash: format specifiers, flags, width and precision, escape sequences, %(fmt)T dates, padded columns, and common errors.

The `printf` command prints text through a format string. Specifiers such as `%s` and `%d` mark where each argument goes and how it looks. The examples use the Bash builtin, the version that runs when you type `printf` in Bash. `/usr/bin/printf`, `dash`, and `awk` differ in a few places, listed in their own card.

Basic Syntax

The format is reused until every argument is consumed.

`printf FORMAT [ARGUMENTS...]`	Print the arguments formatted by FORMAT
`printf '%s\n' "$var"`	Print a variable followed by a newline
`printf 'Done\n'`	A format alone; `printf` never adds a newline itself
`printf '%s\n' one two three`	Format reused for each argument; prints three lines
`printf '%s=%s\n' a 1 b 2`	Reused two arguments at a time; prints `a=1` and `b=2`
`printf '%s %d\n' x`	Missing arguments print as an empty string or `0`
`printf -v var '%05d' 42`	Store `00042` in `var` instead of printing it
`printf -- '-n\n'`	`--` ends options; needed when the format starts with `-`
`printf '%s\n' "$msg" >&2`	Print to standard error

Format Specifiers

Each specifier takes one argument, except `%%`.

`%s`	String
`%b`	String with backslash escapes expanded
`%q`	String quoted for reuse as shell input
`%c`	First character of the argument
`%d, %i`	Signed decimal integer
`%u`	Unsigned decimal integer; `-1` prints `18446744073709551615`
`%o`	Octal integer; `8` prints `10`
`%x, %X`	Hexadecimal in lowercase or uppercase; `255` prints `ff` or `FF`
`%f`	Decimal number, six places by default; `3.14159` prints `3.141590`
`%e, %E`	Exponential notation; `12345.678` prints `1.234568e+04`
`%g, %G`	`%f` or `%e` style, depending on the exponent; `1234567` prints `1.23457e+06`
`%a, %A`	Hexadecimal floating point; `1` prints `0x1p+0`
`%(fmt)T`	Date and time from epoch seconds
`%%`	Literal `%` sign

Flags

Flags go right after `%`, before the width.

`%-8s`	Left-align in the field; `abc` prints `abc`
`%08.3f`	Pad with zeros; `3.14159` prints `0003.142`
`%+d`	Always show the sign; `5` prints `+5`
`% d`	Space before positive numbers, `-` before negative ones
`%#x, %#o`	Add the `0x` or `0` prefix; `255` prints `0xff`
`%#g`	Keep trailing zeros; `1.5` prints `1.50000`
`%'d`	Group digits by locale; `1,234,567` with `en_US.UTF-8`, `1234567` with `C`
`%-05d`	`-` wins over `0`; pads with spaces on the right

Width and Precision

The full form is `%[flags][width][.precision]specifier`. Width is a minimum; precision depends on the type.

`%10s`	Right-align in 10 columns
`%-10s`	Left-align in 10 columns
`%.3s`	Cut the string to 3 characters; `abcdef` prints `abc`
`%10.3s`	Cut to 3 characters, then right-align in 10 columns
`%.2f`	Two decimal places; `3.14159` prints `3.14`
`%.5d`	At least 5 digits; `42` prints `00042`
`%05d`	Zero-pad to 5 columns; `-42` prints `-0042`
`printf '%*s' 10 abc`	Width taken from the argument before the value
`printf '%.*f' 2 3.14159`	Precision taken from an argument; prints `3.14`
`printf '%*.*f' 10 2 3.14159`	Width and precision both from arguments
`printf '%*s' -10 abc`	A negative `*` width left-aligns

Escape Sequences

Escapes work in the format string and in `%b` arguments. `%s` prints them as typed.

`\n`	Newline
`\t`	Horizontal tab
`\\`	Backslash
`\r`	Carriage return; moves the cursor to the start of the line
`\a`	Alert (terminal bell)
`\b`	Backspace
`\v, \f`	Vertical tab, form feed
`\e`	Escape character for ANSI codes: `printf '\e[1mbold\e[0m\n'`
`\NNN`	Byte with octal value NNN; `\101` prints `A`
`\xHH`	Byte with hex value HH; `\x41` prints `A`
`\uHHHH, \UHHHHHHHH`	Unicode character; `\u00e9` prints `é`
`printf '%b\n' 'one\ctwo'`	`\c` in a `%b` argument stops all output; prints `one`

Numbers and Bases

Convert between bases and character codes without `bc`.

`printf '%d\n' 0xff`	Hex to decimal; prints `255`
`printf '%x\n' 255`	Decimal to hex; prints `ff`
`printf '%o\n' 493`	Decimal to octal; prints `755`
`printf '%d\n' 010`	A leading `0` means octal; prints `8`
`printf '%d\n' "'A"`	Character code; prints `65`
`printf "\\x$(printf %x 65)\n"`	Code to character; prints `A`
`printf '%02x' 10 255 0`	Two-digit hex bytes; prints `0aff00`
`printf '%d.%d.%d.%d\n' 0xC0 0xA8 1 1`	Hex octets to an IP address; prints `192.168.1.1`
`printf '%.0f %.0f\n' 2.5 3.5`	Halves round to the even number; prints `2 4`
`printf '%5.1f%%\n' 99.5`	Percentage with a literal `%` sign

Text Recipes

Common one-liners for repeated characters, arrays, and files.

`printf '%.0s-' {1..20}; echo`	Print – 20 times
`printf -v line '%*s' 40 ''; echo "${line// /=}"`	Build a line of 40 = signs
`printf '%s\n' "${arr[@]}"`	Print each array element on its own line
`printf -v csv '%s,' "${arr[@]}"; echo "${csv%,}"`	Join array elements with commas
`printf 'file-%03d.txt\n' 1 2 10`	Zero-padded names; `file-001.txt` to `file-010.txt`
`printf '%s\n' banana apple | sort`	Send lines to another command
`printf 'line1\nline2\n' > notes.txt`	Write several lines to a file
`printf '%s\n' "$entry" >> notes.txt`	Append a line to a file

Dates and Times

`%(fmt)T` is Bash only. `fmt` takes the same codes as `date +FORMAT`.

`printf '%(%F %T)T\n' -1`	Current date and time; `-1` means now
`printf '%(%F)T\n'`	No argument also means now
`printf '%(%T)T\n' -2`	Time the shell started
`printf '%(%F %T)T\n' 1700000000`	Format epoch seconds
`TZ=Asia/Tokyo printf '%(%F %T)T\n' 0`	Another time zone; prints `1970-01-01 09:00:00`
`printf -v today '%(%F)T' -1`	Store the date without running `date`
`printf '%(%s)T\n' -1`	Current epoch seconds
`printf '[%(%T)T] %s\n' -1 "$msg"`	Timestamped log line

Replacing echo

`printf` prints values such as `-n` or `-e` as data. `echo` reads them as options.

`printf '%s\n' "$var"`	`echo "$var"`
`printf '%s' "$var"`	`echo -n "$var"`
`printf '%b\n' "$var"`	`echo -e "$var"`
`printf '\n'`	`echo` (blank line)
`printf '%s\n' "$a" "$b"`	Two `echo` commands

Tables and Columns

Use the same widths in the header and in the rows.

`printf '%-10s %8s\n' NAME SIZE`	Header with a left-aligned name and a right-aligned size
`printf '%-10s %8d\n' nginx 1024 mysql 20480`	Two rows from one call
`printf '%-10s %6.2f\n' cpu 3.5 mem 42.123`	Fixed decimals keep numbers aligned
`printf '%-*s %s\n' "$width" "$name" "$value"`	Column width from a variable
`printf '%s\t%s\n' "$name" "$value"`	Tab-separated output for `cut` or `awk`
`printf '%-10ls %s\n' "$name" "$value"`	Pad by characters, not bytes (Bash 5.3 and later)

Shell Quoting

`%q` escapes a value so the shell reads it back unchanged.

`printf '%q\n' "$file"`	Quote a value for reuse as shell input
`printf '%q\n' "it's a \$file"`	Prints `it\'s\ a\ \$file`
`printf '%q\n' $'tab\tx'`	Control characters use `$'...'`; prints `$'tab\tx'`
`printf '%q ' rm -f "my file.txt"`	Build a command string; prints `rm -f my\ file.txt`
`ssh server "ls -l $(printf '%q' "$dir")"`	Pass a path with spaces through `ssh`
`printf '%#q\n' "a b"`	Single-quote style; prints `'a b'` (Bash 5.3 and later)
`printf '%s\n' "${var@Q}"`	Same idea with parameter expansion

Other printf Versions

`sudo`, `env`, `xargs`, and `find -exec` run `/usr/bin/printf`. Scripts started with `sh` use the `dash` builtin on Ubuntu and Debian.

`type -a printf`	Lists the builtin first, then `/usr/bin/printf`
`-v var`	Bash only; `/usr/bin/printf` prints `-v` as text and `dash` exits with an error
`%(fmt)T`	Bash only; other versions report an invalid conversion or directive
`%q`	`/usr/bin/printf` prints `'a b'` where Bash prints `a\ b`; `dash` has no `%q`
`\xHH`, `\uHHHH`	`dash` prints them as typed
`printf '-n\n'`	Bash reads `-n` as an option; `/usr/bin/printf` prints it
`awk '{printf "%-8s %5d\n", $1, $2}' file.txt`	`awk` separates arguments with commas and does not reuse the format
`awk 'BEGIN { printf "%c\n", 65 }'`	`awk` turns a number into a character (`A`); Bash `%c` prints `6`

Common Mistakes

Quick checks for common `printf` errors and surprises.

`Output runs into the next prompt`	End the format with `\n`
`printf "$var" mangles % or \`	Pass data as an argument: `printf '%s\n' "$var"`; `50% off` prints `500ff` otherwise
`printf: -n: invalid option`	Use `printf -- '-n\n'` or `printf '%s\n' -n`
`printf: 3.5: invalid number`	`%d` takes integers only; use `%.0f` or cut the decimals with `${n%.*}`
`printf: 08: invalid octal number`	A leading zero means octal; force base 10 with `$((10#08))`
`%s prints \t literally`	Use `%b` for arguments that contain escapes
`%c prints 6 for 65`	`%c` takes the first character; use `printf "\\x$(printf %x 65)"`
`An empty array still prints a blank line`	The format runs once with no arguments; check `${#arr[@]}` first
`Accented text breaks alignment or gets cut`	`%s` width and precision count bytes; use `%ls` in Bash 5.3 and later
`%.2f prints 2.67 for 2.675`	Expected: the binary value of `2.675` is slightly below it
`Numerical result out of range`	Integers stop at `9223372036854775807`; use `bc` for larger values

Related Guides

Use these guides for worked examples and related syntax.

Bash printf Command	Full `printf` guide with examples
echo Command in Linux	Simple output and its options
Bash Cheatsheet	Variables, conditionals, loops, and functions
Bash Parameter Expansion Cheatsheet	Trimming, replacement, and `${var@Q}`
date Cheatsheet	Format codes for `%(fmt)T`
awk Cheatsheet	`printf` inside awk programs

Bash Version Support

Run `echo "$BASH_VERSION"` to check your version. macOS still ships Bash 3.2.

`printf -v var`	3.1
`printf -v 'arr[i]'`	4.1
`%(fmt)T, \u, \U`	4.2
`%(fmt)T with no argument`	4.3
`%ls and %lc in characters, %#q`	5.3