NAME

seq — generate numerical sequence

SYNOPSIS

seq [**-w**|**-f** *ormat*] [**-s** *eparator*] [*from* [*increment*]] *to*

DESCRIPTION

Writes numbers from *from* (default **1**) up (down) to *to*, in increments of *increment* (default **1**), separated by newlines, to the standard output stream. All numbers are *long double*s and must be numbers. *increment* may not be **0**.

Unless an explicit format is specified with **-f**, the default if both *from* and *increment* are finite is "`%f`" with fixed minimal precision equal necessary to recover the numbers, otherwise "`%g`".

Be wary of IEEE Std 754-1985 precision loss when dealing with very large boundaries and small *increment*s!

OPTIONS

-w, **--equal-width** Left-pad numbers with "`0`"s, such that each output line has the same width.

-f, **--format**=*format* Use *format*, which must contain exactly one `printf`(3)-style *double* or *long double* format specifier (**%[L]eEfFgGaA**, including any width/precision/&c. parameters); a literal **%** is obtained with **%%**.

-s, **--separator**=*sep* Separate numbers with *sep* instead of a newline. A newline always follows the final number.

EXAMPLES

```
$ seq 3
1
2
3
$ seq -3
$ seq -1 -1 -3
-1
-2
-3

$ seq -ws"$(printf '\t')" -1 0.782 4
-1.000  -0.218  00.564  01.346  02.128  02.910  03.692

$ seq -f'%+.2f%%' -341e-1 341e-2 341e0 | shuf -n3
+156.86%
-10.23%
+109.12%

$ seq 2e20 200000000000000000002
200000000000000000000
```

SEE ALSO

`printf`(3), `strtold`(3)

STANDARDS

Compatible with the GNU system. A mostly-compatible **seq** exists in NetBSD 3.0 and FreeBSD 9.0, which automatically inverts the *increment* if *from* > *to*, adds **-t** to replace the final newline, parses backslash-escapes in **-sft**, and, most importantly, indiscriminately defaults to "**%g**" — meaning that sufficiently large (**1‘000‘000**) numbers are formatted in exponent style by default.

HISTORY

Appeared in Version 8 AT&T UNIX as `seq`(1):

seq [**−w**] [**−p**picture] [first [incr]] last

With a default format of "**`%.0f`**" (rounded to an integer). **`-p`** induced the desired format from the number provided, in the

[**−**][**0**anything]...[**.**[anything]...]

format, with leading zeroes setting the width. As a noted bug, exponent (the only other recognised floating point format, with **`feEgG`** available) *`picture`*s are not recognised.

Version 10 AT&T UNIX defaults to "**`%g`**", making **`-w`** iterate over all values to determine the width (ignoring it for final values over **1‘000‘000**), and replaces **`-p`** with an unchecked **`-f`**.

Both implementations use *`double`*s and pre-compute the iteration count, which must fit in an *`int`*. Additionally, Version 10 AT&T UNIX introduces a rounding error to that computation, causing *`0.3 0.19 1`* to end at `1.06` instead of `0.87`.

Plan 9 from Bell Labs inherits the Version 10 AT&T UNIX implementation. The iteration count was dropped at some indeterminable point after the fourth edition, at the expense of not terminating for big enough input.

BUGS

Sticking to a hard-line *`long double`* behaviour when all arguments are both integral and in range of a 64-bit integer could be considered a bug, see **EXAMPLES** for the side-effects of this precision loss. However, one could say that a strict IEEE Std 754-1985 reading of (*`2e21 − 2e20`*) / (*`1 −`* (*`1 %`* ε)) = ∞ (with $\varepsilon \approx 9$ at the low end) is the *only* valid interpretation, especially with the inconsistency brought on between something like **`seq`** *`2e20 2e21`* and **`seq`** *`2e20 200000000000000000000.1`* — the former would output 1.8e21 numbers, the latter infinitely many. Most systems would agree; the GNU system wouldn’t, but it also does insane shit to **`seq`** arguments, so who knows; it’s important to not lose the forest for the GTrees. What do the users expect? What are the corner-cases (of the implementation, but more-so the user expectations)?