NAME

jot - print sequential or random data

SYNOPSIS

jot [**-cnr**] [**-b** *word*] [**-w** *word*] [**-s** *string*] [**-p** *precision*] [*reps* [*begin* [*end* [*s*]]]]

DESCRIPTION

The **jot** utility is used to print out increasing, decreasing, random, or redundant data, usually numbers, one per line.

The following options are available:

-r Generate random data instead of the default sequential data.

-b *word*
Just print *word* repetitively.

-w *word*
Print *word* with the generated data appended to it. Octal, hexadecimal, exponential, ASCII, zero padded, and right-adjusted representations are possible by using the appropriate printf(3) conversion specification inside *word*, in which case the data are inserted rather than appended.

-c This is an abbreviation for **-w** *%c*.

-s *string*
Print data separated by *string*. Normally, newlines separate data.

-n Do not print the final newline normally appended to the output.

-p *precision*
Print only as many digits or characters of the data as indicated by the integer *precision*. In the absence of **-p**, the precision is the greater of the precisions of *begin* and *end*. The **-p** option is overridden by whatever appears in a printf(3) conversion following **-w**.

The last four arguments indicate, respectively, the number of data, the lower bound, the upper bound, and the step size or, for random data, the seed. While at least one of them must appear, any of the other three may be omitted, and will be considered as such if given as **-** or as an empty string. Any three of these arguments determines the fourth. If four are specified and the given and computed values of *reps* conflict, the lower value is used. If one or two are specified, defaults are assigned starting with *s*, which assumes a default of 1 (or -1 if *begin* and *end* specify a descending range). Then the default values are

assigned to the leftmost omitted arguments until three arguments are set.

Defaults for the four arguments are, respectively, 100, 1, 100, and 1, except that when random data are requested, the seed, *s*, is picked randomly. The *reps* argument is expected to be an unsigned integer, and if given as zero is taken to be infinite. The *begin* and *end* arguments may be given as real numbers or as characters representing the corresponding value in ASCII. The last argument must be a real number.

Random numbers are obtained through arc4random(3) when no seed is specified, and through random(3) when a seed is given. When **jot** is asked to generate random integers or characters with begin and end values in the range of the random number generator function and no format is specified with one of the **-w**, **-b**, or **-p** options, **jot** will arrange for all the values in the range to appear in the output with an equal probability. In all other cases be careful to ensure that the output format's rounding or truncation will not skew the distribution of output values in an unintended way.

The name **jot** derives in part from **iota**, a function in APL.

Rounding and truncation

The **jot** utility uses double precision floating point arithmetic internally. Before printing a number, it is converted depending on the output format used.

If no output format is specified or the output format is a floating point format ('E', 'G', 'e', 'f', or 'g'), the value is rounded using the printf(3) function, taking into account the requested precision.

If the output format is an integer format ('D', 'O', 'U', 'X', 'c', 'd', 'i', 'o', 'u', or 'x'), the value is converted to an integer value by truncation.

As an illustration, consider the following command:

```
$ jot 6 1 10 0.5
1
2
2
2
3
4
```

By requesting an explicit precision of 1, the values generated before rounding can be seen. The .5 values are rounded down if the integer part is even, up otherwise.

```
$ jot -p 1 6 1 10 0.5
```

```
1.0
1.5
2.0
2.5
3.0
3.5
```

By offsetting the values slightly, the values generated by the following command are always rounded down:

```
$ jot -p 0 6 .9999999999 10 0.5
1
1
2
2
3
3
```

Another way of achieving the same result is to force truncation by specifying an integer format:

```
$ jot -w %d 6 1 10 0.5
```

EXIT STATUS

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

EXAMPLES

The command

```
jot - 1 10
```

prints the integers from 1 to 10, while the command

```
jot 21 -1 1.00
```

prints 21 evenly spaced numbers increasing from -1 to 1. The ASCII character set is generated with

```
jot -c 128 0
```

and the strings xaa through xaz with

```
jot -w xa%c 26 a
```

while 20 random 8-letter strings are produced with

```
jot -r -c 160 a z | rs -g 0 8
```

Infinitely many *yes*'s may be obtained through

```
jot -b yes 0
```

and thirty ed(1) substitution commands applying to lines 2, 7, 12, etc. is the result of

```
jot -w %ds/old/new/ 30 2 - 5
```

The stuttering sequence 9, 9, 8, 8, 7, etc. can be produced by truncating the output precision and a suitable choice of step size, as in

```
jot -w %d - 9.5 0 -.5
```

and a file containing exactly 1024 bytes is created with

```
jot -b x 512 > block
```

Finally, to set tabs four spaces apart starting from column 10 and ending in column 132, use

```
expand -‘jot -s, - 10 132 4‘
```

and to print all lines 80 characters or longer,

```
grep ‘jot -s "" -b . 80‘
```

DIAGNOSTICS

The following diagnostic messages deserve special explanation:

illegal or unsupported format '%s' The requested conversion format specifier for printf(3) was not of the form

```
%[#][ ][{+,-}][0-9]*[.[0-9]*]?
```

where "?" must be one of

```
[l]{d,i,o,u,x}
```

or

```
{c,e,f,g,D,E,G,O,U,X}
```

range error in conversion A value to be printed fell outside the range of the data type associated with the requested output format.

too many conversions More than one conversion format specifier has been supplied, but only one is allowed.

SEE ALSO

ed(1), expand(1), rs(1), seq(1), yes(1), arc4random(3), printf(3), random(3)

HISTORY

The **jot** utility first appeared in 4.2BSD.

AUTHORS

John A. Kunze