Specification of the "sequ" command

Copyright © 2013 Bart Massey
Revision 0, 1 October 2013

This specification describes the "universal sequence" command `sequ`. The `sequ` command is a backward-compatible set of extensions to the UNIX `[seq]` (`http://www.gnu.org/software/coreutils/manual/html_node/seq-invocation.html`) command. There are many implementations of `seq` out there: this specification is built on the `seq` supplied with GNU Coreutils version 8.21.

The `seq` command emits a monotonically increasing sequence of numbers. It is most commonly used in shell scripting:

```
  TOTAL=0
  for i in `seq 1 10`
  do
    TOTAL=`expr $i + $TOTAL`
  done
  echo $TOTAL
```

prints 55 on standard output. The full `sequ` command does this basic counting operation, plus much more.

This specification of `sequ` is in several stages, known as *compliance levels.* Each compliance level adds required functionality to the `sequ` specification. Level 1 compliance is equivalent to the Coreutils `seq` command.

The usual specification language applies to this document: MAY, SHOULD, MUST (and their negations) are used in the standard fashion.

Wherever the specification indicates an error, a conforming `sequ` implementation MUST immediately issue appropriate error message specific to the problem. The implementation then MUST exit, with a status indicating failure to the invoking process or system. On UNIX systems, the error MUST be indicated by exiting with status code 1.

When a conforming `sequ` implementation successfully completes its output, it MUST immediately exit, with a status indicating success to the invoking process or systems. On UNIX systems, success MUST be indicated by exiting with status code 0.

Compliance Level 0

Compliance Level 0 of `sequ` requires absolute minimum functionality. A CL0 `sequ` MUST accept exactly two command-line arguments. Each argument SHOULD be a representation of an

integer value. Any other supplied argument syntax is an error.

If the first integer argument is numerically greater than the second, the `sequ` command MUST emit no output. Otherwise, `sequ` MUST print on its output each of the integers between the first and second argument, inclusive. Each output integer MUST be on a line by itself, that is, a line terminated with an appropriate line terminator for the host environment.

Compliance Level 1

Compliance Level 1 of `sequ` adds the full functionality of GNU Coreutils `seq`. This includes the "--format", "--separator", "--equal-width", "--help" and "--version" arguments (as well as the one-character abbreviations of these), the increment argument, and support for floating-point numbers. The `sequ` initialization and increment arguments are now optional, as per the `seq` spec.

The `sequ` "--format" specifier MAY format floating-point numbers differently than `seq`, but it MUST follow some well-described and reasonable floating-point formatting standard.

Backslash-escapes in the "-s" argument string MUST be processed as in C printf(3).

Compliance Level 2

Compliance Level 2 of `sequ` adds additional convenience arguments for formatting.

The arguments that MUST be accepted are as follows:

- *-W, --words*: Output the sequence as a single space-separated line. Equivalent to "-s ' '".

- *-p, --pad* : Output the sequence with elements padded on the left to be all of equal width: the pad character is given by the single-char pad string . Backslash-escapes in MUST be processed as in C printf(3).

  Note that the "-w" command of level 2 is equivalent to "-p '0'".

- *-P, --pad-spaces*: Output the sequence with elements padded with spaces on the left to be all of equal width. Equivalent to "-p ' '".

Compliance Level 3

Compliance Level 3 of `sequ` adds the ability to have sequences of types other than floating-point numbers.

Specifically, CL3 `sequ` MUST accept as arguments and output as results: arbitrary-precision integers, single lowercase alphabetic (ASCII) letters, single uppercase alphabetic (ASCII) letters, and lowercase or uppercase unsigned Roman Numerals.

The `sequ` command MUST accept a new flag, "--format-word" or "-F", that takes a one-word argument indicating the type of the sequence. The `sequ` command MUST accept the format-word arguments "arabic" (for integers), "floating", "alpha" (for letters), "ALPHA", "roman" or "ROMAN"; the all-uppercase variants indicate uppercase sequences.

The `sequ` command MUST accept limit arguments (*start*, *end*, and *increment*) in the format consistent with the format-word. Arabic limit arguments MAY be "promoted" to Roman Numerals when Roman output is requested. The *increment* argument for alpha formats MUST be arabic. Otherwise, the limit arguments MUST be in the same format as the format-word. When no format-word is given, the format MUST be inferred from the format of the mandatory *end* argument.

Compliance Level 4

Compliance Level 4 of `sequ` adds the ability to number the lines of a textfile presented on the input.

CL4 `sequ` MUST accept the "--number-lines" / "-n" argument. This argument indicates that, rather than outputting the sequence on standard output, `sequ` will act as a filter, numbering lines of a file read from standard input to standard output. Each line "number" will be in the format specified by the "--format-word" argument, or inferred from the *start* or *increment* limit argument if the "--format-word" argument is not supplied. The *end* argument is irrelevant when "--number-lines" is supplied; it MUST NOT be accepted. The separator between the line number and the line may be given by the "--separator" argument, defaulting to space.

Compliance Level 5

Compliance Level 5 of `sequ` adds the ability to infer a sequence from a given prefix.

As an alternative to the limit arguments of previous Compliance Levels, CL5 `sequ` may accept a sequence specifier of the form:

value [*value*] [*value*] ... ".." *value*

When the ".." argument is present, the non-flag arguments MUST be parsed in *inference mode*.

In inference mode, `sequ` picks a best match for the *pattern* (partial sequence of values leading up to the ".."), and then continues the sequence until the *end* value (after the "..") is succeeded.