PV286 Project Assignment

February 26, 2023

Panbyte – pandoc, but for bytes

A tool for conversions between various representations of byte sequences.

Your task is to implement a tool that can load data provided in one of the `FORMATS` and transform it into a specified `FORMAT`, which it outputs. The tool should be able to handle multiple inputs at the same time that are divided by a given `DELIMITER`. The tool should support specifying input file and output file, yet stdin and stdout are the default options.

In case an invalid input is provided, output an error and terminate with a non-zero return code. Make sure that the program is able to handle such inputs securely.

This specification does not aim to be complete. In case you discover an undescribed situation, try to come up with a reasonable behavior. If in doubt, contact your project feedback person.

CLI

Provide the following command-line interface.

```
./panbyte [ARGS...]

ARGS:
  -f FORMAT     --from=FORMAT           Set input data format
                --from-options=OPTIONS  Set input options
  -t FORMAT     --to=FORMAT             Set output data format
                --to-options=OPTIONS    Set output options
  -i FILE       --input=FILE            Set input file (default stdin)
  -o FILE       --output=FILE           Set output file (default stdout)
  -d DELIMITER  --delimiter=DELIMITER   Record delimiter (default newline)
  -h            --help                  Print help

FORMATS:
  bytes                                 Raw bytes
  hex                                   Hex-encoded string
  int                                   Integer
  bits                                  0,1-represented bits
  array                                 Byte array
```

If you get multiple lines (or sequences divided by a different `DELIMITER`), process them independently and join the results by the same delimiter.

Supported formats

Each format type can be used as input or output. Format type can optionally have options that influence how it should be processed. In case contradictory options are provided, use the latter. In case multiple options can be combined, apply all of them.

`bytes` – Raw bytes

Raw bytes as received/output by the program.

When used as input, consider delimiter only if provided explicitely.

Examples

```
$ echo test | ./panbyte -f bytes -t bytes
test
```

`hex` – Hex-encoded byte string

Bytes encoded as a hexadecimal string.

Input may be interleaved with whitespaces, ignore them.

Examples

```
$ echo 74657374 | ./panbyte -f hex -t bytes
test
$ echo test | ./panbyte -f bytes -t hex
74657374
$ echo 74 65 73 74 | ./panbyte -f hex -t bytes
test
```

`int` – Integer

Unsigned integer representation of underlying bytes.

Input options

- `big` – Store the integer in big-endian representation (most significant byte at the lowest addres; default).
- `little` – Store the integer in little-endian representation (least significant byte at the lowest address).

Output options

- `big` – Interpret bytes as an an integer in big-endian representation (most significant byte at the lowest addres; default).
- `little` – Interpret bytes as an integer in little-endian representation (least significant byte at the lowest address).

Examples

```
$ echo 1234567890 | ./panbyte -f int -t hex
499602d2
$ echo 1234567890 | ./panbyte -f int --from-options=big -t hex
499602d2
```

```
$ echo 1234567890 | ./panbyte -f int --from-options=little -t hex
d2029649
$ echo 499602d2 | ./panbyte -f hex -t int
1234567890
$ echo 499602d2 | ./panbyte -f hex -t int --to-options=big
1234567890
$ echo d2029649 | ./panbyte -f hex -t int --to-options=little
1234567890
```

`bits` – Bits

String of 0 and 1 characters, representing a sequence of bits. If needed, pad the string with zeros.

Input may be interleaved with whitespaces, ignore them.

Input options

- `left` – If necessary, pad input with zero bits from left (default).
- `right` – If necessary, pad input with zero bits from right.

Examples

```
$ echo 100 1111  0100 1011 | ./panbyte -f bits -t bytes
OK
$ echo 100111101001011 | ./panbyte -f bits --from-options=left -t bytes
OK
$ echo 100111101001011 | ./panbyte -f bits --from-options=right -t hex
9e96
$ echo OK | ./panbyte -f bytes -t bits
0100111101001011
```

`array` – Byte array

Byte array as represented in programming languages.

Inputs may contain mixed values (covering all permitted array values – see below) and arbitrary whitespace characters between them; opening and closing brackets have to match.

Outputs have to be formatted as per options.

When both `--from` and `--to` options are `array`, support processing of nested arrays. I.e., if you are given a nested array on input, keep the brackets in the same places, but transform them according to output options.

Output options

- `0x` – Represent bytes as a 0x-prefixed hex number (e.g., `0xff`; default).
- `0` – Represent bytes as a decimal number (e.g., `255`).
- `a` – Represent bytes as characters (e.g., `'a'`, `'\x00'`).
- `0b` – Represent bytes as 0b-prefixed binary number (e.g., `0b11111111`).
- `{` or `}` or `{}` – Use curly brackets in output (default).
- `[` or `]` or `[]` – Use square brackets in output.
- `(` or `)` or `()` – Use regular brackets in output.

Examples

```
$ echo 01020304 | ./panbyte -f hex -t array
{0x1, 0x2, 0x3, 0x4}
$ echo "{0x01, 2, 0b11, '\x04'}" | ./panbyte -f array -t hex
01020304
$ echo "{0x01,2,0b11 ,'\x04' }" | ./panbyte -f array -t array
{0x1, 0x2, 0x3, 0x4}
$ echo "[0x01, 2, 0b11, '\x04']" | ./panbyte -f array -t array --to-options=0x
{0x1, 0x2, 0x3, 0x4}
$ echo "(0x01, 2, 0b11, '\x04')" | ./panbyte -f array -t array --to-options=0
{1, 2, 3, 4}
$ echo "{0x01, 2, 0b11, '\x04'}" | ./panbyte -f array -t array --to-options=a
{'\x01', '\x02', '\x03', '\x04'}
$ echo "[0x01, 2, 0b11, '\x04']" | ./panbyte -f array -t array --to-options=0b
{0b1, 0b10, 0b11, 0b100}
$ echo "(0x01, 2, 0b11, '\x04')" | ./panbyte -f array -t array --to-options="("
(0x1, 0x2, 0x3, 0x4)
$ echo "{0x01, 2, 0b11, '\x04'}" | ./panbyte -f array -t array --to-options=0 \
  --to-options="["
[1, 2, 3, 4]
```

Examples with nesting (applicable only in `array-array` conversions):

```
$ echo "[[1, 2], [3, 4], [5, 6]]" | ./panbyte -f array -t array
{{0x1, 0x2}, {0x3, 0x4}, {0x5, 0x6}}
$ echo "[[1, 2], [3, 4], [5, 6]]" | ./panbyte -f array -t array \
  --to-options="{" --to-options=0
{{1, 2}, {3, 4}, {5, 6}}
$ echo "{{0x01, (2), [3, 0b100, 0x05], '\x06'}}" | ./panbyte -f array -t array \
  --to-options=0 --to-options="["
[[1, [2], [3, 4, 5], 6]]
$ echo "()" | ./panbyte -f array -t array
{}
$ echo "([],{})" | ./panbyte -f array -t array --to-options="["
[[], []]
```