bbe

binary block editor
Version 0.2.2

by Timo Savinen

This file documents version 0.2.2 of `bbe`, a binary block editor.

1 Preliminary information

The `bbe` program is a sed-like editor for binary files. `bbe` performs basic byte related transformations on blocks of input stream. `bbe` is non-interactive command line tool and can be used as a part of a pipeline. `bbe` makes only one pass over input stream.

`bbe` contains also grep-like features, like printing the filename, offset and block number.

2 Samples using bbe

Few examples of running bbe:

'bbe -b "/\x80\x50\x0e/:12" -e "d 0 3" -e "c BCD ASC" -e "A \x0a" -e "w /tmp/numbers" -o /dev/null /tmp/bdata'

Task here is to extract BCD coded numbers from the file /tmp/bdata and write them in ascii format with newline to file /tmp/numbers. 12 bytes long blocks containing the BCD-numbers start with three byte sequence of values '0x80', '0x50' and '0x0e'. First three bytes (the block start sequence) are removed ('d 0 3') rest of the block is transformed from BCD to Ascii ('c BCD ASC') and a newline character is appended at the end of the block ('A \x0a'). All transformed blocks are written to /tmp/numbers ('w /tmp/numbers'). Nothing is written to the output ('-o /dev/null').

'bbe -b ":525" -e "i 524 \x0a" -o /tmp/data_with_nl /tmp/data'

A newline is added after every 525'th byte of the file /tmp/data. Data with newlines is written to /tmp/data_with_nl.

'bbe -b ":526" -e "d 525 1" -o /tmp/data /tmp/data_with_nl'

Every 526'th byte (inserted newline in previous example) is removed from the file /tmp/data_with_nl. Data without newlines is written to /tmp/data.

'bbe -e "s/\x0d\x0a/\x0a/"'

Same effect as has command dos2unix.

3 How to run bbe

bbe accepts several commands to operate on blocks. Commands are executed in the same order as they appear in command line or in a script file. Order is significant, because the changes made to current byte by previous commands are seen by next commands.

3.1 Program invocation

The format for running the bbe program is:

```
bbe option ...
```

bbe supports the following options:

`-b` *BLOCK*
`--block=`*BLOCK*
: Block definition.

`-e` *COMMAND*
`--expression=`*COMMAND*
: Add command(s) to the commands to be executed. Commands must be separated by semicolon.

`-f` *script-file*
`--file=`*script-file*
: Add commands from *script-file* to the commands to be executed.

`-o` *file*
`--output=`*file*
: Write output to *file* instead of standard output.

`-s`
`--suppress`
: Suppress printing of normal output, print only block contents.

`-?`
`--help`
: Print an informative help message describing the options and then exit successfully.

`-V`
`--version`
: Print the version number of bbe and then exit successfully.

All remaining options are names of input files, if no input files are specified or - is given, then the standard input is read.

3.2 Block definition

bbe devides the input stream to blocks defined by the -b option. If block is not defined, the whole input stream is considered as one block. Commands have effect only inside a block, rest of the input stream remains untouched. Currently bbe supports only one block definition per invocation. If input stream consists of different blocks, several bbes can be combined in a pipeline.

A block can be defined several ways:

`N:M` Block starts at offset N of input stream (first byte is 0). Block is M bytes long. This definition allows only one block to be defined.

`:M` The whole input stream is divided to M-size blocks.

`/start/:/stop/`
Blocks start with sequence *start* and end with sequence *stop*. Both *start* and *stop* are included to blocks.

`/start/:` Blocks start with sequence *start* and ends at next occurrence of *start*. Only the first *start* is included to block.

`:/stop/` Blocks start at the beginning of input stream or after the end of previous block. Block ends at first occurrence of *stop*. Only the last *stop* is included to blocks.

It is possible to use c-like byte values in N, M, *start* and *stop*. Values in *start* and *stop* must be escaped with \, \ can be escaped as \\.

Byte values can be expressed in decimal, octal or hexadecimal e.g. in *start* and *stop*:

`\123, \32 or \0`
Decimal values

`\012, \08 or \0278`
Octal values

`\x0a, \x67 or \xff`
Hexadecimal values

Also escape code `\y` can be used. Decimal values of `\y`'s:

`\a` 7

`\b` 8

`\t` 9

`\n` 10

`\v` 11

`\f` 12

`\r` 13

`\;` 59
Semicolon must be escaped, because it is a command delimitter.

Values of N and M can be given in decimal, octal and hexadecimal:

`123, 32 or 112232`
Decimal values

`0128, 08123 or 0`
Octal values

`x456a, x167 or xffde`
Hexadecimal values

3.3 bbe commands

Commands in bbe can be divided in two groups: Block related commands and byte related commands. Block related commands operate at block level e.g. remove a block. Byte related commands work allways inside a block and they don't have effect beyond the block boundaries.

Same escape codes for byte values in *string*s can be used as in *start* and *stop* of block definition.

Block commands are:

`I` *`string`* Write the *string* to output stream before the block.

`D [`*`N`*`]` Delete the *N*'th block. If *N* is not defined all blocks are deleted from output stream. **Note**: First block is number one.

`A` *`string`* Write the *string* to output stream after the block.

`J` *`N`* Commands appearing after this command have no effect until *N* blocks are found. Means "Jump first *N* blocks". **Note**: Commands that are defined before this command have effect on every block.

`L` *`N`* Commands appearing after this command have no effect after *N* blocks are found. Means "Leave blocks after *N*'th block". **Note**: Commands that are defined before this command have effect on every block.

`N` Before block contents the file name where the current block starts is printed with colon.

`F` *`f`* Before block contents the current stream offset and colon is printed in format specified by *f*. Stream offset starts at zero. *f* can have one of following values:

- *H* Hexadecimal.
- *D* Decimal.
- *O* Octal.

`B` *`f`* Before block contents the current block number and colon is printed in format specified by *f*. Block numbering starts at one. *f* can have one of the sames codes as `F`-command.

`> file` Before printing a block, the contents of file `file` is printed.

`< file` After printing a block, the contents of file `file` is printed.

Byte commands are:

Note: The *n* in byte commands is offset from the beginning of current block, first byte is number zero.

`c` *`from to`* Converts bytes from *from* to *to*. Currently supported formats are:

- `ASC` Ascii
- `BCD` Binary Coded Decimal

Note: Bytes, that cannot be converted are passed through as they are. e.g. in ASC -> BCD conversion, ASCII characters not in range `'0'` – `'9'` are not converted.

`d` *n* *m*`|*` Delete *m* bytes starting from the offset *n*. If * is defined instead of *m*, then all bytes of the block starting from *n* are deleted.

`i` *n* *string*
Insert *string* after byte number *n*.

`j` *n* Commands appearing after `j`-command have no effect concerning bytes 0-*n* of the block.

`l` *n* Commands appearing after `l`-command have no effect concerning bytes starting from the byte number *n* of the block.

`u` *n* *c* All bytes from start of the block to offset *n* are replaced by *c*.

`f` *n* *c* All bytes starting from offset *n* to the end of the block are replaced by *c*.

`p` *format* Contents of block is printed in formats specified by *format*. *format* can contain following format codes:

H	Hexadecimal.
D	Decimal.
O	Octal.
A	Ascii, nonprintable characters are printed as space.
B	Binary.

format can contain several codes, values are then separated by hyphen.

`r` *n* *string*
Replace bytes with *string* starting at the byte number *n* of the block.

`s/`*search*`/`*replace*`/`
All occurences of *search* are replaced by *replace*. *replace* can be empty. Separator `/` can be replaced by any character not present in *search* or *replace*.

`w file` Contents of blocks are written to file `file`. **Note**: Data inserted by commands `A`, `I`, `>` and `<` are written to file `file` and `j` and `l` commands have no effect on `w`-commands. Zero size files are not preserved.
Filename can contain format string `%B` or `%nB`, these format strings are replace by current block number (starting from one), causing every block to have it's own file. In `%nB`, the `n` is field width in range 0-99. If `n` has a leading zero, then the block numbers will be left padded with zeroes.

`y/`*source*`/`*dest*`/`
Translate bytes in *source* to the corresponding bytes in *dest*. *source* and *dest* must have equal length. Separator `/` can be replaced by any character not present in *source* or *dest*.

`&` *c* Performs binary and with *c* on block contents.

`|` *c* Performs binary or with *c* on block contents.

`^ c` Performs exclusive or with *c* on block contents.

`~` Performs binary negation on block contents.

`x` Exchange the contents of nibbles (half an octet) of bytes.

3.4 Limitations

At least in GNU/Linux `bbe` should be able to handle big files (> 4 GB), other systems are not tested.

There are however, some limitations in block and command definitions:

Strings in block definition
Search string in `s` *command*
: are limited to *16384* bytes.

4 How bbe works

bbe scans the input stream just once, so the last block may differ from the block definition, because bbe doesn't 'peek' the end of the input stream. Last block may be shorter than defined, e.g. if block is defined as '/string/:128' and if the end of input stream is found before 128'th byte of the last block is reached, the last block remains shorter.

Basic execution cycle:

1. Start of the block is searched. If found, data before block is written to output stream (unless -s is defined) and step 2 is executed.
2. Block commands affecting the start of the block (I, D, J, N, F, > and B) are executed.
3. The block is scanned byte by byte and all byte commands (lower case letters) are executed. **Note**: Commands are executed on results of previous commands, if e.g. the first byte of the block is deleted, the following commands don't 'see' the removed byte.
4. When end of the block is reached the end of the block commands (A and <) are executed.
5. Next block is searched, data between the blocks, if not suppressed with -s, is written to output stream.

Few examples:

'echo "The quick brown fox jumps over a lazy dog" | bbe -b "/The/:21" -e "j 4" -e "s/ /X/"'

Output is

```
The quickXbrownXfoxXjumps over a lazy dog
```

The only block in this is

```
The quick brown fox j
```

All spaces in the block are converted to X's, before conversion first 4 bytes are skipped.

'echo "The quick brown fox jumps over a lazy dog" | bbe -b ":/ /" -e "J 1" -e "A \x0a"'

Output is:

```
The quick
brown
fox
jumps
over
a
lazy
dog
```

All blocks end at space, a newline character is inserted after every block except the first block.

'echo "The quick brown fox jumps over a lazy dog" | bbe -e "r 4 fast\x20" -e "s/f/c/"'

Output is:

```
The cast  brown cox jumps over a lazy dog
```

Also the f in fast is converted to c.

'echo "1234567890" | bbe -b ":1" -e "L 9" -e "A -"'

Output is

```
1-2-3-4-5-6-7-8-9-0
```

Hyphen is inserted after every 1 byte long block,but not after 9'th block.

'bbe -s -b "/First line/:/Last line/" /tmp/text'

Print lines between sentences 'First line' and 'Last line'.

'bbe -s -b "%<a %:%%" -e "s/\x0a/ /" -e "A \n" ./index.html'

Extract all links from ./index.html. To get one link per line, all newlines are converted to spaces and newline is added after every link.

'bbe -b "/\x5f\x28\x02/:10" -s -e "F d" -e "p h" -e "A \n" ./bindata'

10 bytes long sequences starting with values x5f x28 and x02 are printed as hex values. Also the file offset is printed before each sequence and new line is added after every sequence. Example output:

```
52688:x5f x28 x02 x32 x36 x5f x81 x64 x01 x93
68898:x5f x28 x02 x39 x46 x5f x81 x64 x41 x05
69194:x5f x28 x02 x42 x36 x5f x81 x64 x41 x05
```

'bbe -b "/Linux/:5" -s -e "N;D;A \x0a" /bin/* | uniq'

Print the file names of those programs in /bin directory which contains word 'Linux'. Example output:

```
/bin/loadkeys:
/bin/mkbimage:
/bin/ps:
/bin/uname:
```

'bbe -b "/\x5f\x81\x18\x06/:10" -s -e "B d;d 0 4;c BCD ASC;A \n" ./bindata'

Print BCD numbers and their block numbers in ascii format. Numbers start with sequence x5f x81 x18 x06. The start sequence is not printed.

'bbe -b "/\x5f/:2" -e "j 1;& \xf0" -o newdata bindata'

The least significant nybble of bytes after x5f is cleared.

‘`bbe -b "/\xff\xd8\xff/:/\xff\xd9/" -s -e "w pic%02B.jpg" -o /dev/null manual.pdf`’

Extract jpg-images from pdf-file to separate jpg-files (assuming that the jpg start/stop sequences does not appear in other context than jpg-images). Files will be named as `pic01.jpg`, `pic02.jpg`, `pic03.jpg`,. . .

‘`bbe -b "_<body>_:_</body>_" -s -o temp nicebody.html`’
‘`bbe -b "_<body>_:_</body>_" -e "D;< temp" -o tmpindex.html index.html`’
‘`mv tmpindex.html index.html`’

The body part of the html-document `index.html` is replaced by the body of the document `nicebody.html`.

5 Reporting Bugs

If you find a bug in `bbe`, please send electronic mail to `tjsa@iki.fi`. Include the version number, which you can find by running '`bbe --version`'. Also include in your message the output that the program produced and the output you expected.

If you have other questions, comments or suggestions about `bbe`, contact the author via electronic mail to `tjsa@iki.fi`. The author will try to help you out, although he may not have time to fix your problems.

Table of Contents