lowdown — simple markdown translator

Kristaps Dzonsons

2021-09-23

1 lowdown — simple markdown translator

lowdown is a Markdown translator producing HTML5, *roff* documents in the **ms** and **man** formats, LaTeX, gemini, OpenDocument, and terminal output. The open source C source code has no dependencies.

The tools are documented in lowdown(1) and lowdown-diff(1), the language in lowdown(5), and the library interface in lowdown(3).

To get and use *lowdown*, check if it's available from your system's package manager. If not, download, verify, and unpack the source. Then build:

```
% ./configure
% make
% make regress
# make install install_libs
```

lowdown is a BSD.lv project. Its portability to OpenBSD, NetBSD, FreeBSD, Mac OS X, Linux (glibc and musl), Solaris, and IllumOS is enabled by oconfigure and checked by BSD.lv's build system.

One major difference between *lowdown* and other Markdown formatters it that it internally converts to an AST instead of directly formatting output. This enables some semantic analysis of the content such as with the difference engine, which shows the difference between two markdown trees in markdown.

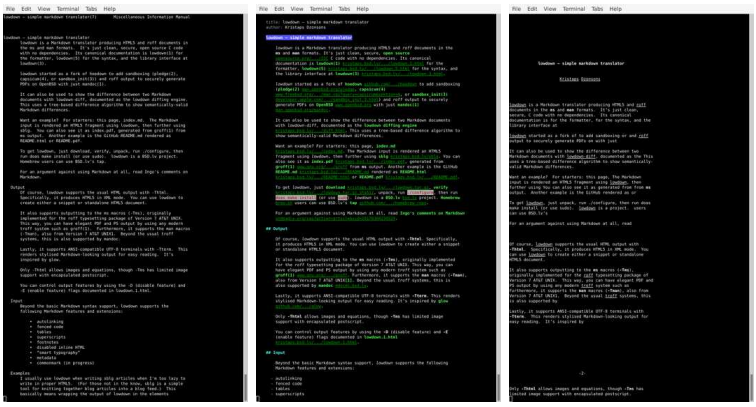
1.1 Output

lowdown produces HTML5 output in XML mode with **-thtml**, LaTeX documents with **-tlatex**, "flat" OpenDocument XML documentx (OpenDocument version 1.3) with **-tfodt**, Gemini with **-tgemini**, *roff* documents with **-tms** and **-tman**[1] outputs (via groff or mandoc, or directly on ANSI terminals with **-tterm**.

[1] You may be tempted to write manpages in Markdown, but please don't: use mdoc(7), instead — it's built for that purpose! The **man** output is for technical documentation only (section 7).

The **-tlatex** and **-tms** are commonly used for PDF documents, **-tman** for manpages, **-thtml** or **-tgemini** for web, and **-tterm** for the command line.

By way of example: this page, index.md, renders as index.latex.pdf with LaTeX (via **-tlatex**), index.mandoc.pdf with mandoc (via **-tman**), or index.nroff.pdf with groff (via **-tms**).



-tman -tterm -tms

Only **-thtml** and **-tlatex** allow images and equations, though **-tms** has limited image support with encapsulated postscript.

1.2 Input

Beyond traditional Markdown syntax support, *lowdown* supports the following Markdown features and extensions:

- autolinking
- fenced code
- tables
- superscripts (traditional and GFM)
- footnotes
- disabled inline HTML
- “smart typography”
- metadata
- commonmark (**in progress**)
- definition lists
- extended attributes
- task lists
- admonitions

1.3 Examples

Want to quickly review your Markdown in a terminal window?

```
lowdown -tterm README.md | less -R
```

I usually use *lowdown* when writing sblg articles when I'm too lazy to write in proper HTML5. (sblg is a simple tool for knitting together blog articles into a blog feed.) This basically means wrapping the output of *lowdown* in the elements indicating a blog article. I do this in my Makefiles:

```
.md.xml:
     ( echo "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" ; \
       echo "<article data-sblg-article=\"1\">" ; \
       echo "<header>" ; \
       echo "<h1>" ; \
       lowdown -X title $< ; \
       echo "</h1>" ; \
       echo "<aside>" ; \
       lowdown -X htmlaside $< ; \
       echo "</aside>" ; \
       echo "</header>" ; \
       lowdown $< ; \
       echo "</article>" ; ) >$@
```

Note: you'll want to make sure that the title and aside are properly HTML formatted, as **-X** will not escape for the output format.

If you just want a straight-up HTML5 file, use standalone mode:

```
lowdown -s -o README.html README.md
```

This can use the document's meta-data to populate the title, CSS file, and so on.

The troff output modes work well to make PS or PDF files, although they will omit equations and only use local PS/EPS images in **-tms** mode. The extra groff arguments in the following invocation are for UTF-8 processing (**-k**), tables (**-t**), and clickable links and a table of contents (**-mspdf**).

If outputting PDF, use the pdfroff script instead of **-Tpdf** output. This allows image generation to work properly. If not, a blank square will be output in places of your images.

```
lowdown -stms README.md | groff -itk -mspdf > README.ps
lowdown -stms README.md | pdfroff -itk -mspdf > README.pdf
```

The same can be effected with systems using mandoc:

```
lowdown -stman README.md | mandoc -Tps > README.ps
lowdown -stman README.md | mandoc -Tpdf > README.pdf
```

More support for PDF (and other print formats) is available with the **-tlatex** output.

```
lowdown -stlatex README.md | pdflatex
```

For terminal output, troff or mandoc may be used in their respective **-Tutf8** or **-Tascii** modes. Alternatively, *lowdown* can render directly to ANSI terminals with UTF-8 support:

```
lowdown -tterm README.md | less -R
```

Read lowdown(1) for details on running the system.

1.4 Library

lowdown is also available as a library, lowdown(3). This is what's used internally by lowdown(1) and lowdown-diff(1).

1.5 Testing

The canonical Markdown tests are available as part of a regression framework within the system. Just use `make regress` to run these and many other tests.

If you have valgrind installed, `make valgrind` will run all regression tests with all output modes and store any leaks or bad behaviour. These are output to the screen at the conclusion of all tests.

I've extensively run AFL against the compiled sources with no failures—definitely a credit to the hoedown authors (and those from whom they forked their own sources). I'll also regularly run the system through valgrind, also without issue. The afl/in directory contains a series of small input files that may be used in longer AFL runs.

1.6 Code layout

The code is neatly layed out and heavily documented internally.

First, start in library.c. (The main.c file is just a caller to the library interface.) Both the renderer (which renders the parsed document contents in the output format) and the document (which generates the parse AST) are initialised.

The parse is started in document.c. It is preceded by meta-data parsing, if applicable, which occurs before document parsing but after the BOM. The document is parsed into an AST (abstract syntax tree) that describes the

document as a tree of nodes, each node corresponding an input token. Once the entire tree has been generated, the AST is passed into the front-end renderers, which construct output depth-first.

There are a variety of renderers supported: html.c for HTML5 output, nroff.c for **-ms** and **-man** output, latex.c for LaTeX, gemini.c for Gemini, odt.c for OpenDocument, term.c for terminal output, and a debugging renderer tree.c.

1.7 Installing

You'll need a C compiler with essential build tools (make, cc, etc.). First, configure the system:

```
./configure
```

You can pass variables like `PREFIX` and such here. To install the binaries, run:

```
make install
```

For libraries, you can additionally run:

```
make install_libs
```

This may be split into `install_shared` and `install_static` for shared and static libraries, respectively.

1.8 Example

For example, consider the following:

```
## Hello **world**
```

First, the outer block (the subsection) would begin parsing. The parser would then step into the subcomponent: the header contents. It would then render the subcomponents in order: first the regular text "Hello", then a bold section. The bold section would be its own subcomponent with its own regular text child, "world".

When run through the **-Ttree** output, it would generate:

```
LOWDOWN_ROOT
  LOWDOWN_DOC_HEADER
  LOWDOWN_HEADER
    LOWDOWN_NORMAL_TEXT
      data: 6 Bytes: Hello
    LOWDOWN_DOUBLE_EMPHASIS
      LOWDOWN_NORMAL_TEXT
        data: 5 Bytes: world
```

This tree would then be passed into a front-end, such as the HTML5 front-end with **-thtml**. The nodes would be appended into a buffer, which would then be passed back into the subsection parser. It would paste the buffer into `<h2>` blocks (in HTML5) or a `.SH` block (troff outputs).

Finally, the subsection block would be fitted into whatever context it was invoked within.

1.9 Compatibility

lowdown is fully compatible with the original Markdown syntax as checked by the Markdown test suite, last version 1.0.3. This suite is available as part of the `make regress` functionality.

1.10 How Can You Help?

Want to hack on *lowdown*? Of course you do.

- Using a perfect hash (such as **gperf**) for entities.
- There are bits and bobs remaining to be fixed or implemented. You can always just search for `TODO`, `XXX`, or `FIXME` in the source code. This is your best bet.
- Footnotes in **-tms** with groff extensions should use pdfmark to link to and from the definition.
- If you want a larger project, a **-tpdf** seems most interesting (and quite difficult given that UTF-8 need be present). Another project that has been implemented elsewhere is a parser for mathematics such that `eqn` or similar may be output.