Flexible Formatting with Linuxdoc-SGML

Christian Schwarz

July 1995
Reprinted with permission of *Linux Journal*

Abstract

Have your cake and eat it too with this simple but powerful text processing facility assembled by a well-known Linux guru.

As Linux becomes more and more popular, a lot of documentation is required, not only for newcomers, but for all users. Just think of all the FAQs and HOWTOs, manual pages, and the books of the linux documentation project that everyone needs for the daily work. Some people want to read these documents as plain ASCII text, while others want to read them over the World Wide Web, or perhaps just print them on their postscript printer. It is possible to take an ASCII document and make an HTML version of it for the Web, and a nicely-formated postscript version for people to print, but then all the different formats would have to be maintained separately, which is theoretically possible, but in real life doesn't happen.

We need a documentation system that is capable of producing different formats from a single source.

The Linux Documentation Project faced this exact dilemma when the HOWTO project was started, and so Matt Welsh wrote the Linuxdoc-SGML package to solve this exact problem. With this package, all documentation gets a similar layout. But since the way SGML works is very flexible, you can use the system to write many different kinds of documentation: as an example, the XFree86 project uses Linuxdoc-SGML for all of its own documentation.

An example

```
<!doctype linuxdoc system>

<article>

<title>The Very Short Story
<author>A. Author
<date>1 Jan 1970

<p>
Once upon a time, they lived happily ever after.
</article>
```

As you can see here, the Linuxdoc-SGML syntax is very simple. Commands are written in angle brackets like `<command>`. When they apply to a certain block of the text they appear in a pair surrounding that block, so `<article>` before the block is balanced by `</article>` after the block. There is also an abbreviation for

the latter case if the block is short: `<tt/typewriter font/` is equivalent to `<tt>typewriter font</tt>`; both make the enclosed text be in a monospaced (constant-width) font.
The first line of the document specifies the document type. Here you will always specify `linuxdoc system`, since this refers to the main macros of the Linuxdoc-SGML package. Then you start your document with the `<article>` command and close it at the end with the corresponding "article off" `</article>` command. The article itself starts with the title, the author, and the date (which is optional).
After that you can start writing text. The `<p>` command indicates the beginning of the first paragraph. You don't have to worry about spaces or line breaks when writing the text, since multiple spaces between words are ignored and line breaks are inserted at the appropriate positions automatically. If you want to begin a new paragraph, you can just insert a blank line, which is a "synonym" for `<p>`.

Running Linuxdoc-SGML

Linuxdoc-SGML is actually a collection of programs that work together to provide the final output. You need to know how to use each of them: several examples will help. The `format` program creates files designed for LaTeX, groff, or makeinfo, and is part of the process of creating HTML files, which is explained more fully below. The `-T` argument tells `format` which program it is writing files for.
There is one utility for running each of the formatting programs (groff, etc). Each has a name starting with 'q' like 'qtex'.
To get a PostScript file via LaTeX, just type

```
format -T latex example.sgml > example.tex
qtex example
```

and Linuxdoc-SGML will first create a LaTeX-format file, then use LaTeX to process that file, and then use `dvips` to turn it into the PostScript file `example.ps`. Note that you need to have LaTeX and dvips installed along with Linuxdoc-SGML in order for this to work.
If you prefer a DVI file, you may add a `-d` switch to `qtex` command:

```
format -T latex example.sgml | qtex -d > example.dvi
```

The plain ASCII output is created with a similar procedure. Just run

```
format -T nroff example.sgml | qroff > example.txt
```

To get texinfo output that can be read with the GNU info program, just run

```
format -T info example.sgml
```

This will create the necessary files in the current directory automatically. Of course, you need the GNU texinfo package installed on your system to make texinfo files.
The HTML output needs a little bit more care since two compilation stages are necessary to get all cross references built up right. First, you have to have the `LINUXDOC` environment variable set up correctly; you will want to put a line like

```
export LINUXDOC=~/linuxdoc-sgml-1.2
```

in your bash startup file, or

```
setenv LINUXDOC=~/linuxdoc-sgml-1.2
```

in your csh or tcsh startup file.
Once that is working, you have to run several commands in order to get finished HTML:

```
format -T html example.sgml | prehtml | fixref > tmp.html
format -T html example.sgml | prehtml >> tmp.html
cat tmp.html | html2html example > example.html
rm tmp.html
```

It's a good idea to put these commands in a shell script since you will call these commands very often. Here's a simple version you can use:

```
#!/bin/bash
# sgml2html
[ -z "$1" ] && { echo "What file?"; exit 1 }
BASE=`basename $1 .sgml`
[ ! -f $BASE.sgml ] && { echo "No file $BASE.sgml"; exit 2 }
TMP=$$tmp.html
format -T html $1 | prehtml | fixref > $TMP
format -T html $1 | prehtml >> $TMP
cat $TMP | html2html $BASE > $BASE.html
rm $TMP
```

This script requires, that your input file has the extension `.sgml`.
All of this is documented more completely in the excellent short manual provided with Linuxdoc-SGML.

The details

Having explained the necessary "overhead" we can start writing larger documents. Therefore, you will probably want to create sections and subsections, start the document with an abstract, and insert a table of contents. You can just add the following lines after the `date` line:

```
<abstract>
Here is the abstract.
</abstract>

<toc>

<sect>The First Section

<sect1>The First Subsection
```

Depending on the output format you want to create, the table of contents will have different styles: in a LaTeX document you get a standard table of contents and in an html file you will get a list of cross references for the different sections and subsections.
As you can see from our example, the `<sect>` command creates a new section and the `<sect1>` command a new subsection. You can access five different levels simply by increasing the number in the command name. Note that you will still have to insert the `<p>` command to start the first paragraph after an sectioning command.
For technical documentation it is often necessary to include "verbatim" text: text this is passed through without interpretation by the SGML parser. You can do this with the `<verb>` command:

```
<verb>This text is not interpreted </verb>
```

However, some very special characters do need separate handling even when they appear inside a verbatim environment. For example, the sequence of an opening angle bracket and a slash (`</`) has to be written as `&etago;`. The manual that comes with the Linuxdoc-SGML package contains a list of all special

characters and the corresponding command names, to make them appear literally in the output instead of being interpreted as commands.
The Linuxdoc-SGML package supports three different font styles beside the normal font: boldface, italics, and typewriter. To select a font just insert the command `bf`, `em`, or `tt`, respectively. To switch back to the default font just use the corresponding "slashed" command, as in the following example:

```
This <em>text</em> is written in italics!
```

There is also an abbreviated version of each of these which is sometimes useful:

```
This <em/text/ is written in italics!
```

Now let's take a look at different list types, that are supported:

- `itemize` for bulleted lists, as demonstrated in this list
- `enum` for numbered lists, and
- `descrip` for description lists.

You just start the list "environment" with the `<list>` command and close it with `</list>`. To produce a new item (either a bullet or a number) you can use the `<item>` command. In the description environment, you have to use the `<tag>` command with an argument that contains the "keyword", as in the following example:

```
<descrip>
<tag/First./ This is the first point.
<tag>Second.</tag> And this is the second.
</descrip>
```

If you want to create WWW pages with this package, you may want to create cross references to other WWW pages. This is done with the `<url>` command, as in the following example:

```
<url url="http://gnus.com/pub/text.html" name="Text document">
```

This creates a link to the WWW page `text.html` on the specified host and displays `Text document` as the title of the link. When you create non-HTML documents, both the name and the URL are shown, so that the reader can still find the document.

Getting Linuxdoc-SGML

The newest version of the Linuxdoc-SGML package is 1.2 and is available on sunsite.unc.edu in the directory `/pub/linux/utils/text`.
It is written by Matt Welsh with the help of a few others and is currently maintained by Greg Hankins. The package includes manual and a tutorial, which are both very good examples of what can be done with Linuxdoc-SGML package, since they are written in Linuxdoc-SGML format.

Christian Schwarz studies mathematics in Munich and has worked with linux for two years. He contributed the texinfo support for the Linuxdoc-SGML package. You may reach him at the address `schwarz@monet.m.isar.de`.