Howtos with LinuxDoc

Table of Contents

Howtos with LinuxDoc

David S. Lawyer

v0.09, November 2007

This is about how to write HOWTOs using the simple LinuxDoc markup. It's primarily for Linux Documentation Project authors (and future fledging authors who want to get started fast). If you want to use the more difficult and complex DocBook markup (including XML) see the LDP Authoring Guide.

1. Introduction

1.1 If you want to start immediately

To only learn LinuxDoc, skip to Learning LinuxDoc. If you want to start writing immediately, then you may try a "fill in the blanks" template which will generate LinuxDoc formatted output. The LDP HOWTO Generator. You may use this to just start writing your Howto and then finish it later by using a text editor on your PC.

1.2 Copyright and License

1.3 Should you write a HOWTO ?

Do you know things about Linux for which no good free documentation is available and which would be useful to others? Even if you don't know the subject well, you can still write about it if you're eager, willing, and able to learn more about it and have the time to do it. Can you write clearly using a word processor or editor? Do you want to help thousands of others and let them read what you write at no cost to them? Once you've written a document, are you willing to receive email suggestions from readers and selectively use this info for improving your HOWTO? Would you like to have your writings be available on hundreds of websites throughout the world? If you can answer yes to these, then you're encouraged to write something for the Linux Documentation Project (LDP). But be warned that it may take more time than you expected.

1.4 Why I wrote this

Why did I write this when there is already an "LDP Authoring Guide"? Well, the LDP guide is a long and detailed work. If you want to get started quickly, you need something much simpler and shorter.

Thanks to Matt Welsh for his example.sgml file which I used as a major source of info for the example sections.

2. Information on Writing a HOWTO

2.1 Copyright

All HOWTOs and other LDP documents are copyright by the authors so the LDP doesn't have any special rights to your writing. We only accept documents that have a license which permits anyone to copy and distribute the document. We encourage authors to also allow modification in their license. This way, if the author stops maintaining a document, someone else can do so. For more details see our Manifesto.

2.2 Choosing a topic

If you are not sure what to write about, take look at some of LDP's documents including the ones in Unmaintained HOWTOs. Pick a topic you're interested in that needs good documentation. If you find something already written and maintained that needs improvement, try to contact the author, first and make suggestions. If you can't reach the author, look at the license and see if you're allowed to modify the document. But even in the cases where you are not allowed to make modifications and improvements, you can just write a new document on the same topic from scratch, using a new outline new sources of information.

3. The Format of HOWTOs

3.1 Introduction

Our HOWTOs are released to the public in various formats: Plain Text, HTML, PostScript, and PDF. Instead of having to write the same HOWTO in all of these formats, just one HOWTO is written in a source format, DocBook or LinuxDoc, which gets converted by computer into all of the other formats.

To get an idea of what a source format looks like, take a look at the source file of a webpage (if you haven't already). You will see all sorts of words in <angle brackets>. These are called tags. These webpages (tags and all) are in html: *Hypertext Markup Language*. The LDP uses formats something like this for its documents.

The markup languages LDP uses meet the requirements of either *Standard Generalized Markup Language (SGML)* or *XML*. The LDP now uses the following two flavors of sgml: *LinuxDoc* and *DocBook* as well as the *DocBook* flavor of XML. Interestingly, it turns out that html is just another flavor of sgml (but some features people use in html violate sgml rules so it's not pure sgml anymore).

This mini-HOWTO is all about using the simple LinuxDoc flavor of sgml. You may call it "LinuxDoc markup". It can be converted by computer to html, plain text, postscript, pdf, and DocBook. It's a lot easier than HTML or DocBook and you don't need a special editor for it as it's easy to type in the tags (or use macros for them) using your favorite editor or word processor.

4. Comparing LinuxDoc to DocBook

Before reading this section it's best to have at least an elementary knowledge of tags in markup languages. So if you don't know much about this, you may first want to look at Example 1 for LinuxDoc markup.

One way to compare is to inspect real HOWTOs at the LDP site. Click on either DocBook Index or LinuxDoc Index to find them. You'll notice that the DocBook docs are cluttered with tags while the LinuxDoc docs have

far fewer tags and are thus more readable and easier to write and modify. Some documents at LDP are marked up much more clearly than others so you should inspect at least a few documents.

You may at first think that DocBook is more advanced since there are so many more tags but this isn't necessarily so. If you create a document in LinuxDoc and convert it to DocBook (by computer) there will be many more tags in the DocBook version, including new tags that were not in the LinuxDoc version. Why? Because LinuxDoc permits the omission of tags while DocBook doesn't. And in this way LinuxDoc is more flexible and advanced than DocBook. Not only does LinuxDoc often allow the omission of end-tags like </title> but allows the omission of both start and end tag pairs. For example, in LinuxDoc paragraphs are normally separated by blank lines instead of pairs of paragraph tags <p> (<para> in DocBook) so that paragraph tags are seldom needed. Another example is just after the start of a new section of a LinuxDoc document. You just type in the title of the section after the <sect> tag while DocBook requires one to also enclose the title in a pair of <title> tags.

When one runs a program to convert LinuxDoc to say HTML, the first thing the program does is to find all the omitted tags and add them to the document. For example, a pair of tags are added which are equivalent to the pair of <title> tags after a <sect> tag (including the cases of <sect1> tags, etc.). So in a sense, LinuxDoc also has a lot of tags, but they are hidden from the user to make the document both much easier to write and read. In most cases, the author writing a document in LinuxDoc is unaware of the existence of most of the missing tags. There's no need (in most cases) for the author to know about them since they exist only in the computer memory (or a temporary file) when the PC is converting LinuxDoc to some other format (like to HTML). Well, it is possible to save a file showing the added tags but that is done mainly by programmers debugging the LinuxDoc computer code (or by people like me that are curious about how it works).

The DocBook tags are often longer than the equivalent LinuxDoc tags such as instead of <emphasis>. For a short comparison see the author's website Comparison of DocBook to LinuxDoc

However, the fact that many tags are omitted when writing LinuxDoc (but are found by the software) is only one reason why LinuxDoc documents have fewer tags. Another reason is that DocBook actually has many more tags to use than LinuxDoc does. For example, LinuxDoc has just one tag for the author's name while DocBook has separate tags for the first middle and last names. So in this sense DocBook is more powerful than LinuxDoc but in another sense it's weaker since it can't find missing tags. The solution to this dilemma would be to merge LinuxDoc with DocBook so as to retain the advantages of each. DocBook would then need to abandon it's current status as a "xml" language since xml prohibits the omission of tags, etc. while sgml allows it.

One reason why omitted tags are not allowed in DocBook is that it makes it easier for programmers to write software to parse it and convert it to other formats. There is no need for software capable of finding and adding missing tags. But making it easier for programmers, makes it more difficult for the much large number of writers that have to use DocBook.

There are special editors or word processors like Lyx and Bluefish that make it easier to type DocBook documents. For example, Bluefish automatically adds end tags. But for people who don't want to learn a new editor or word processor, LinuxDoc is a lot easier since one can type in the tags by hand or create a set of macros to insert the (<emphasis> tags. I use the vim editor and if I type ;s it inserts the <sect1> tag. ;r does <sect>, ;i does <item>, etc. For the header tags on the first several lines of the document, I just copy them from another document and change the words after the tags. Of course I don't need to change the author's name or email. So a major advantage of LinuxDoc is that one can easily use it with the same editor or word processor that they currently use and know.

Some people who don't understand the situation have advocated making DocBook about as easy as LinuxDoc by just using a subset of the DocBook tags. This doesn't work because DocBook will still require a few times as many tags to get the same results due to the requirement of DocBook to not permit any omitted tags.

In spite of the advantages of LinuxDoc, the number of people who use DocBook greatly exceeds the number using LinuxDoc partly because Linuxdoc was promoted by book publishers. There's a program by Reuben Thomas (ld2db) which can convert LinuxDoc documents to DocBook. It's not perfect and you will likely need to do some manual editing. The LDP also automatically converts a LinuxDoc HOWTO to DocBook after you submit it. It would be nice to have another program to automatically convert DocBook docs to LinuxDoc so they can be modified by people using the simpler LinuxDoc markup.

5. Learning LinuxDoc

5.1 Introduction

LinuxDoc is a lot easier to learn than DocBook. But most of what you learn about LinuxDoc would also be useful for DocBook. So if you eventually decide to go for DocBook, most of the effort spent on learning LinuxDoc will not be wasted.

One way to learn it is by examples. I've written 3 example files ranging from easy to intermediate. The contents of these files have been copied into this Howto. To turn them into individual files you may cut them out (start with the first tag) and write them to files. Then you could try turning one into text by using say sgml2txt --pass="-P-cbou" some-example.sgml to see what it looks like. Make sure the sgml file names end in .sgml.

If you want to look at some real examples you can just go to an LDP mirror site, find the HOWTOs and select LinuxDoc SGML. Or go to the main site directly: Howto Index (linuxdoc) Now for the first simple example.

5.2 Example 1 (file name: example1.sgml)

```
<!doctype linuxdoc system>
<article>
<title>First Example (example1)
<author>David S.Lawyer

<sect> Introduction
<p> This is a very simple example of "source" for the LinuxDoc text
formatting system.  This paragraph begins with a paragraph tag (a "p"
enclosed in angle brackets).  Notice that there are other tags, also
enclosed in angle brackets.  If you don't see any tags, then you are
reading a converted file so find the source file: example1.sgml (which
contains the tags).

This is the next paragraph.  Note that it is separated from the above
paragraph by just a blank line.  Thus it needs no "p" tag in front of
it.  The "p" tag is only needed for the first paragraph of a section
(just after the sect-tag).  The file suffix: sgml stands for Standard
Generalized Markup Language.  You are now reading the LinuxDoc flavor
of sgml as specified in the very first line of this file.

<sect> Tags
<p> Tags are words inside angle brackets.  The "sect" tag above
marks the start of a new section of this example document.
"Introduction" was the first section and you are now reading the
```

```
second section titled "Tags".  If this were a long document (like a
book), a section would correspond to a chapter.

Note that there are "article", "title" and "author" tags at the start
of this article.  At the end of this article is an "/article" tag
marking the end of this article.  Thus there is a pair of "article"
tags, the first being the start tag and the second being the end tag.
Thus this entire article is enclosed in a pair of "article" tags.  In
later examples you'll see that there are other tags that come in pairs
like this.   They affect whatever is between the pairs (start tag and
end tag).  Any tag name which has "/" just before it is an "end tag".

When this source code is converted to another format (such as plain
text using the program sgml2txt) the tags are removed.  Tags only help
the sgml2txt program make the conversion.  There are more tags to
learn.  So once you understand this example1, please go on to the next
example: example2.  You don't need to actually memorize the tags, as
they will be repeated (but with little or no explanation) in later
examples.
</article>
```

5.3 Example 2 (file name: example2.sgml)

```
<!-- This is a comment.  It's ignored when this source file gets
converted to other formats.  -->
<!-- The tag below says that this file is in LinuxDoc format -->
<!doctype linuxdoc system>

<article>

<title>Second Example (example2)
<author>David S. Lawyer
<date>v1.0, July 2000

<abstract>
This is the abstract.  This document is the second example of using
the Linuxdoc-SGML flavor of sgml.  It's more complex than the first
example (example1.sgml) but simpler than the third example
(example3.sgml).  After you digest this you'll be able to write a
simple HOWTO using LinuxDoc.  End of the abstract.
</abstract>

<!-- The "toc" = Table of Contents.   It will be created here. -->
<toc>

<!-- Begin the main part of the article (or document) here.  The part
above this is sort of a long header. -->

<sect>This Second Example (example2.sgml)

<p>Unless you're familiar with markup languages, you should first
read example1.sgml.  You may want to run these example files thru a
translator such as sgml2txt to convert them to text and notice how the
result looks different than this "source" document with all its tags.

<sect>Article Layout
<sect1> Document Body

<p> After the header comes the body of the document, consisting of
nested sections marked by sect-tags.  Subsections are
marked by sect1-tags.  Since this is the first subsection
```

```
within the 2nd main section, it's becomes section 2.1.  Within a
subsection marked by sect1 there may be sub-subsections like
sect2.  There are even tags like sect3, sect4, etc., but you are
unlikely to need them.  Note the the real tags must be enclosed in
angle brackets < and >.

<sect2> This is a sub-sub-section
<p>
It's 2.1.1.  Note that a "p" tag may be on a line by itself.  This
doesn't change a thing in the resulting documents.

<sect1>Document Header
<p> One way to create a header part is just to copy one from another
.sgml file.  Then replace everything except the tags with the correct
info for your document.  This is like using a "template".

<sect> More Features in example3
<p> With the tags in this example2 you can write a simple short document
a few pages long.  But for longer documents or for other important
features such as putting links into documents, you need to study the
next example: example3.  It will also show you how to create lists and
fonts.
</article>
```

5.4 Example 3

```
<!doctype linuxdoc system>
<!-- Note the mailto: after my name.  This allows the reader of html
format to click on my email address to send me email -->

<article>
<title>Third Example (example3)
<author>David S. Lawyer <url url="mailto:dave@lafn.org">
<date>v1.0, July 2000
<abstract>
This document is the third example of using the LinuxDoc flavor of sgml.
It's more complex than the second example.
</abstract>
<!-- Comment: toc = Table of Contents  -->
<toc>

<sect> Fonts
<p>
While they will not show up in a plain text output, they will work
for other conversions.
<bf>boldface font</bf>   <em>emphasis font</em>     <sf>sans serif</sf>
<sl>slanted font</sl>    <tt>typewriter font</tt>   <it>italics font</it>
There's another way to get these same fonts by enclosing the text in
slashes like this: <bf/boldface font/    <em/emphasis font/
<sf/sans serif/ <sl/slanted font/     <tt/typewriter font/
<it/italics font/ Note that DocBook doesn't have font tags so it may
be best not to use fonts if you plan to convert to DocBook.

<sect> Links <label id="links_">
<p> You may create links (something that html browsers may click on to
go somewhere else).  They might just go to another part of this
document (cross-references) such as to the "label" above, or they
could go to a website on the Internet.

<sect1> Cross-References
<p> If you click on <ref id="links_" name="Links"> you will be taken to
```

```
the start of the "Links" section above (which is labeled links_).
The label id may be any word you choose but it's a good idea to avoid
common words so that you can search for unique labels using your
editor.  That's why I use links_ (with the underline).  The name of
this link will be shown (in html format) as the name to click on.
This name (Links) will also be present in the text rendition.

<sect1> URL Links
<p> If you click on <url url="http://www.tldp.org"> you will get
to the Linux Documentation Project website.  The next link adds a name
which people will click on: <url url="http://www.tldp.org"
name="Linux Documentation Project">.  Using this second method, you may
not even need to explain where the link leads to since it's obvious by
the name.

<sect> Prohibited Characters
<p> Any word you type between angle brackets will be interpreted as a
tag.  But what if you want to display a tag in a document?  For this
you use a code word for the angle characters.

You may use &lt for < and &gt for >.  lt = Less Than, gt =
Greater Than.  For example, here's a p-tag: &lt;p&gt;.  Of
course it doesn't actually start any paragraph, but it will appear in
the converted document as <p>.  These codes all start with an &
character.  The ; after the lt is to separate it.  It's not needed if
there is a space after it.  For example: 3 &lt 4.  Actually, if
you knew that its OK to use an unpaired > then you could have written
<p> as &lt;p>.  This will not be mistakenly recognized as a tag
since there is no opening <.  Actually 3 < 4 works fine too.

There are other characters that you can't put into the document text
directly.  For & in an AT modem command use: AT&amp;.  If other
characters cause you trouble (they seldom will) see <ref id="ch_codes"
name="Character Codes (macros)"> or the "guide" that comes with
linuxdoc-tools or sgml-tools.

<sect> Verbatim, Code & Newline
<sect1> Verbatim
<p> If you want to insure that it will look exactly like you typed it
after it's converted  to other formats, use verbatim (verb).  This is
useful for creating tables, etc.  But some things still get recognized
as markup even though they are between verbatim tags.  This includes
the macros starting with & and end tags with /.

<tscreen><verb>
% sgml2txt --pass="-P-cbou" example.sgml
</verb></tscreen>
The "tscreen" sets the font to typewriter and indents it nicely.

<sect1> Code
<p> This encloses computer code between two dashed lines.
<tscreen><code>
Put computer source code here
</code></tscreen>

<sect1> Newline
<p> To force a newline use <newline>
This sentence always starts at the left margin.

<sect>Lists
<p>
This puts items into a list with a bullet at the start of each item.
```

```
They start with the "itemize" tag.
<itemize>
<item> This is the first item in a list.
<item> This is the second item
       <itemize>
       <item> Multiple levels (nesting) are supported.
       <item> The second item in this sublist
       </itemize>
       <enum>
       <item> Enumerated lists using <tt/enum/ also work.
       <item> This is item number 2
       </enum>
<item> The final item in the main list
</itemize>
</article>
```

5.5 LinuxDoc Quick Reference Sheet

Header Part

```
<!doctype linuxdoc system>
<article>
<title>Quick Reference Sheet
<author>David S. Lawyer
<date>v1.0, July 2000
<abstract> abstract here </abstract>
<toc> <!-- Comment: toc = Table of Contents  -->
```

Body Layout

```
<sect> Chapter 1            Note: Put a <p> on the first line of
<sect1> Subsection 1.1      each section (or subsection, etc.)
<sect1> Subsection 1.2
<sect> Chapter 2            Choose title names to replace "Chapter"
<sect1> Subsection 2.1      "Subsection", etc.
<sect2> Sub-subsection 2.1.1
<sect2> Sub-subsection 2.1.2
<sect1> Subsection 2.2
</article>
```

Fonts

There are two ways to get these:

```
<bf>boldface font</bf>   <em>emphasis font</em>     <sf>sans serif font</sf>
<sl>slanted font</sl>    <tt>typewriter font</tt>   <it>italics font</it>
<bf/boldface/    <em/emphasis/     <sf/sans serif/
<sl/slanted/     <tt/typewriter/   <it/italics/
```

Lists (nesting is OK)

```
Ordinary unnumbered list:              Numbered list:
<itemize>                              <enum>
<item> First item                      <item> First item
<item> Second item                     <item> Second item
<item> etc.                            <item> etc.
</itemize>                             </enum>
```

Links

```
   Cross-References:                          An Email Link:
<ref id="links_" name="Links">          <url url="mailto:bob@tldp.org">
```

Newline, Verbatim, URLs

```
To force a newline <newline>
<tscreen><verb>
<url url="http://www.tldp.org">
<url url="http://www.tldp.org" name="Linux Documentation Project">.
</verb></tscreen>
```

Character Codes (macros)

You don't always need to use these.

- Use `&` for the ampersand (&),
- Use `<` for a left bracket (<),
- Use `>` for a right bracket (>),
- Use `&etago;` for a left bracket with a slash (</)

Use of these are optional and I seldom use them.

- Use ``` `` ``` and `''` for opening and closing double quotes
- Use `&shy;` for a soft hyphen (that is, an indication that this is a good place to break a very long word to insert a hyphen for horizontal justification).

Only use these if LinuxDoc complains about it or fails to generate them in the formated document. I've seldom had to use them.

- Use `&dollar;` for a dollar sign ($),
- Use `&num;` for a hash (#),
- Use `&percnt;` for a percent (%),
- Use `&tilde;` for a tilde (~),
- Use `&dquot;` for ".

6. Getting/Using the LinuxDoc Software

You could write a LinuxDoc document without having any LinuxDoc software. However, it's likely that it would contain some errors in the tags (or their use) so that it would be returned to you for correction. Even if there were no errors, the results might not not look quite right. So it's best for you to have the software to convert your source code on your computer.

The Debian distribution of Linux has a linuxdoc-tools package. There is also a rpm package for non-Debian distributions. It was formerly called sgml-tools. Don't use the sgmltools-2 package which is primarily for DocBook-sgml.

To use linuxdoc-tools you run converter programs on the *.sgml files. For example for versions after 0.9.21-0.8 to get text output, type: "sgml2txt -f --blanks=1 my-HOWTO.sgml". For earlier versions due to a bug you must substitute --pass="-P-cbou" for -f. (If interested, see Old Problem of Escape Sequences in Text

Output for more info on this bug.) To get html output,type: "sgml2html my-HOWTO.sgml". If it shows errors, it will show the line number and the column number where the error is in the source file. Typing "man -k sgml" should show you a number of other programs with a one-line description of each but not all of them are for linuxdoc-sgml.

7. Errors and Error Messages

7.1 Errors That Don't Create Error Messages

A major error is to forget to put a <p>. tag after a section heading. Then there is no end to the section heading and all the following paragraphs become part of the section heading and get into the table of contents. Linuxdoc should be improved to spot this error. To spot it manually, look at the table of contents in html or text format. Fixing it requires just adding the missing p-tag.

7.2 Actual Error Messages

When you are running the linuxdoc program (sgml2html or sgml2txt for example) you are likely to see some error messages. You need to edit your document and fix the errors.

Omission of closing quotes is a common error. If you get an error message that makes no sense and notice that at the error location are some quotes (" ") that are OK, then the likely error is that you have previously typed an opening quote with no closing quote. Like: <.... id="my home page > so linuxdoc thinks that the next quote, which may be many paragraphs after the missing quote location, is the closing quote. Then after the false closing quote it expects a > to finish the tag but doesn't find one. To find and fix the missing quote, just search backwards for a ".

A single error can cause a lot of error messages. In the example above, a number of tags may be inside erroneous quotes that shouldn't have been inside any quotes. So the linuxdoc will not find these tags. As a result, it will not know that a certain tag is open and if it finds a closing tag it will tell you that that tag was not open. For example, if the <itemize> tag was missed, then <item> tags may make no sense to linuxdoc and it will report an error for each such tag found beyond the false closing quote.

One thing you should know is that the tag names are not case sensitive so the error messages will show tag names in upper case (capital letters) even though you typed them in lower case.

To understand the error messages better requires an understanding of the jargon on sgml which you don't really need to learn unless you get errors which you can't seem to get fixed and you don't understand the error message. The following sector is about such jargon.

8. Jargon in Error Messages

8.1 Introduction

You really shouldn't need to read this section unless either you're either having problems or you're curious about how sgml and linuxdoc work. Error messages may contain words like "element", "entities, "attribute", "literal", and delimiter". Various elements, entities and attributes are defined for linuxdoc in the "Data Type Definition" (or dtd) for LinuxDoc. The dtd doesn't define them in sentences but uses a rather cryptic format to define their syntax (but not their semantics).

8.2 Elements

An "element" is something like a tag. But it's a much broader concept. Elements exist not only in linuxdoc but in all sgml languages like say html. Your entire document is partitioned into elements. But elements are nested, which is to say that some elements may occur within other elements. If you use the <article> tag for your document, then all of the document is the <article> element, except for the very first tag which says that what follows is linuxdoc. And within this article element are nested many other elements.

For example, each paragraph is an element, even though the paragraphs are separated from each other by blank lines instead of tags. But there's an implicit tag surrounding each paragraph and the software that parses a linuxdoc writing will actually insert these missing tags. It will also insert end tags (closing tags) where you didn't need to write any. In this way, linuxdoc saves you a lot of time. So an element will consist of a start tag and the end tag (for this start tag) and everything in between (often including other elements and their tags). Note that the tags omitted but they still are implicitly there. In some cases, a tag doesn't enclose anything, like the url tag for a link to the internet. Such tags are themselves elements. Within the article-element are found sect-elements (sections) starting with <sect>. Then within sect-elements are often found sect1-elements (subsections), etc.

There are few cases where an element occurs but the use of both start and end tags are optional. So even if you have no such tags in your document, the parts of the document that they should have enclosed are still elements for the missing tags.

An entity is like a macro definition. For example, one could define the name "list" to mean the various types of lists. Then this name list is used only in the dtd to specify, for example, that a list may occur within a paragraph. It's just a shorthand for the writer of a dtd. This kind of an entity is never used in a linuxdoc document. But there's also another type of entity that can be used inside a document and that's one that defines a special character such as amp;etago for </ (end tag open). You would use this when you want to, for example, put </article> in the middle of a sentence to explain what it means so that the software that converts LinuxDoc doesn't think it's really at the end of the article.

8.3 Literals and Delimiters

A "literal" is a name of something, like the name one clicks on in an html link. It may be one or more words long. A delimiter is what separates something from something else. For a "quote" the last " is the closing delimiter. So for name="my website" the literal is 'my website' and the delimiters of this literal are the two " marks, the first " an opening delimiter and second " a closing delimiter. So if a "literal is missing a closing delimiter it means that you neglected to put an ending " after a name.

9. Writing the HOWTO

9.1 Before you start writing

First join the discuss list by going to www.en.tldp.org and submit your proposal to this list. If you're taking over an unmaintained HOWTO, contact the former author. This may required by the copyright-license but you should do it out of courtesy even it it's not required.

9.2 Guidelines

These are mostly by Tim Bynum (a former HOWTO coordinator).

- Be sure and use an accepted format (such as LinuxDoc :-).
- Try to use meaningful structure and organization, and write clearly. Remember that many of the people reading HOWTOs do not speak English as their first language.
- Make sure that all of the information is correct. I can't stress this enough. When in doubt, speculate, but make it clear that you're only guessing. I use ?? if I'm not sure.
- Make sure that you are covering the most recent version of the available software.
- Consider including a "FAQ" section or sections called "Common Problems" or "Trouble Shooting".
- Be sure to copyright it in your name and include a license which meets the requirements stated in the LDP manifesto.
- Use the standard header with title, author, and date (including a version number). See Example 3
- Lastly, be prepared to receive email questions and comments from readers. How much you help people is up to you but you should make use of good suggestions and reports of errors. You may also get some "thank you for writing this" email.

9.3 Submitting the HOWTO, etc.

After you have written the HOWTO, email the SGML source to submit@tldp.org. Then all you need to do is to keep the HOWTO up-to date by submitting periodic updates to the same email as you used for the first edition.

10. More Information

There's a HOWTO: Linuxdoc-Reference that covers it in much greater detail than this mini-HOWTO.

11. Appendix

11.1 Old Problem of Escape Sequences in Text Output

Prior to version 0.9.21-0.8 there was a bug for the case of text output. For sgml2txt, the option --pass="-P-cbou" was needed to get pure text output since otherwise if you used the -f option, you got text output which put emphasis on words and letters by the use of escape sequences and overstriking. An example of a bullet made by overstriking is +^Ho which on a printer would type +, then backspace (^H), and then type o over the existing +. This doesn't seem to work on display terminals (they can't overstrike). Note that even if you have the most recent version, you'll still get this unwanted output if you fail to use the -f option.

In case you are interested, the --pass passes the -P-cbou option to the groff program (used by sgml2txt) and the -P option of groff passes the -cbou options to grotty (a post-processor for groff) forcing grotty to generate just plain text output. See the grotty man page. In brief: -c avoids escape sequences but allows overstrikes but -bou prohibits overstrikes when the -c option is used. The result is no overstrikes and no escape sequences in the output. -b prohibits overstrikes to make a character look bold; -u prevents overstrikes for underlining; and -o prohibits other kinds of overstrikes like the bullet example above. An alternate way to eliminate overstrikes is to use the -f option with sgml2txt but you still have to pass the -c option to grotty to eliminate escape sequences unless you have the newer version.

What a mess it was! The default should probably be plain text so that all of this passing of options wouldn't be needed. I've finally got them to fix this so after about mid-2007 you can use just the -f option instead of --pass="-P-cbou". If you get these escape sequences and overstrikes in your output file but use the Linux "cat" command to display the text, it looks great. But using pagers or editors on the text output file usually results in the escape characters being eaten so you see a bunch of unwanted characters in your text that were supposed to be part of escape sequences. In some cases, pagers can display certain overstrikes OK but editors (like vim) don't. So eliminating all overstrikes permits you to use any editor or pager to read it.