HOWTO−HOWTO

Table of Contents

Table of Contents

HOWTO−HOWTO

Mark Komarinski

List the tools, procedures, and hints to get HOWTO authors up to speed and writing.

Table of Contents

Introduction

History

This document was started on Aug 26, 1999 by Mark F. Komarinski after two day's worth of frustration getting tools to work. If even one LDP author is helped by this, then I did my job.

New versions

The newest version of this can be found on my homepage http://www.cgipc.com/Ümarkk/in its SGML source. Other versions may be found in different formats at the LDP homepage http://www.linuxdoc.org/.

Comments

Comments on this HOWTO may be directed to the author (markk@linuxdoc.org).

Version History

v1.8 (May 17, 2000)

- Corrected errors in LinuxDoc section that didn't render properly.
- Added content about Cygnus tools − thanks to Dan Scott
- Added OpenJade information and link
- More DocBook rendering information − thanks to Greg Ferguson

Copyrights and Trademarks

(c) 1999−2000 Mark F. Komarinski

This manual may be reproduced in whole or in part, without fee, subject to the following restrictions:

- The copyright notice above and this permission notice must be preserved complete on all complete or partial copies

Acknowledgements and Thanks

Thanks to everyone that gave comments as I was writing this. This includes Deb Richardson and Daniel Barlow and other members of the ldp−discuss list. Some sections I got from the HOWTO Index (available at many LDP locations) and the sgmltools documentation. There are pointers to sgmltools and the LDP elsewhere in this document. The sections on network access to CVS was partially written by Serek (ser@serek.arch.pwr.wroc.pl). Sections on DocBook were written by Jorge Godoy (godoy@conectiva.com.br). A great deal of thanks to both of them for their help.

Background on the LDP and SGML

The LDP

The Linux Documentation Project (LDP) was started to provide new users a way of getting information quickly about a particular subject. It not only contains a series of books on administration, networking, and programming, but has a large number of smaller works on individual subjects, written by those who have used it. If you want to find out about printing, you get the Printing HOWTO. If you want to do find out if your Ethernet card works with Linux, grab the Ethernet HOWTO, and so on. At first, many of these works were in text or HTML. As time went on, there had to be a better way of managing these documents. One that would let you read it from a web page, a text file on a CD−ROM, or even your hand−held PDA. The answer, as it turns out, is SGML.

SGML

The Standard Generalized Markup Language (SGML) is a language that is based on embedding codes within a document. In this way, its similar to HTML, but there is where any similarities end. The power of SGML is that unlike WYSIWYG (What You See Is What You Get), you don't define things like colors, or font sizes, or even some kinds of formatting. Instead, you define elements (paragraph, section, numbered list) and let the SGML processor and the end program worry about placement, colors, fonts, and so on. HTML does the same thing, and is actually a subset of SGML. SGML has really two parts that make it up. First is the Structure, which is what is commonly called the DTD, or Document Type Definition. The DTD defines the relationship between each of the elements. The DocBook DTD, used to create this document, is an example of this. The DTD gives a common look and feel to each document that's created using the DTD. Second is the Content, which is what gets rendered by the SGML processor and is eventually seen by the user. This paragraph is content, but so would a graphic image, table, numbered list, and so on. Content is surrounded by tags to separate out each element.

Why SGML instead of HTML or other formats?

SGML provides for more than just formatting. You can automatically build indexes, table of contents, and links within the document or to outside. The Jade and OpenJade packages also lets you export (I'll call it render from here on) SGML to LaTeX, info, text, HTML, and RTF. From these basic formats, you can then create other formats such as MS Word, PostScript, PDF and so on. Programs like LyX allow you to write in TeX format, then export it as SGML and render from SGML to whatever you chose. In the end, SGML is more concerned about the way elements work instead of the way they look. A big distinction, and one that will let you write faster, since you don't have to worry about placement of paragraphs, font sizes, font types, and so on.

The tools

In this section, I'll go over some of the tools that you'll need or want to use to create your own LDP documentation. I'll describe them here, and better define them later on, along with how to install them. If you use some other tool to assist in writing LDP, please let me know and I'll add a blurb here for it.

Jade

Required − ftp://ftp.jclark.com/pub/jade/jade−1.2.1.tar.gz

Jade is the front−end processor for SGML. It uses the DSSSL and DocBook DTD to perform the verification and rendering from SGML into the target format.

OpenJade

Replacement for Jade − http://openjade.sourceforge.net/

An extension of Jade written by the DSSSL community (see below for what a DSSSL is). Some applications require jade, but are being updated to support either software package.

DSSSL

Required − http://nwalsh.com/docbook/dsssl/db152.zip

The Document Style Semantics and Specification Language tells jade how to render a SGML document into print or online form. The DSSSL is what converts a title tag into an <H1> tag in HTML, or bold, 14 point Times Roman for RTF, for example. Documentation for DSSSL is located at http://nwalsh.com/docbook/dsssl/db152d.zip. Note that modifying the DSSSL doesn't modify DocBook itself. It merely changes the way the rendered text looks. The LDP uses a modified DSSSL that provides for a table of contents.

DocBook DTD (version 3.1)

Required − http://www.oasis−open.org/docbook/sgml/3.1/docbk31.zip

The DocBook DTD defines the tags and structure of a DocBook SGML document. Modifying the DTD, such as adding a new tag, doesn't make it DocBook anymore.

sgmltools−lite

Highly Recommended − http://sgmltools−lite.sourceforge.net/

This is the successor to the sgmltools project, which has officially been disbanded for over a year. Since then, Cees de Groot has created a slightly different project, which acts as a wrapper to the jade SGML processor. It hides much of the ugliness of syntax. This author was able to install the old sgmltools package followed by the sgmltools−lite and could format this document quite easily. There's even a man page for sgmltools showing syntax.

TeX

Optional

TeX (rhymes with blech!) is the markup language of choice for many, including those in the mathematics world. I still remember many Calculus exams that were written in TeX. It is also one of the first markup languages that is still around (the other being the *roff formats used in man pages). TeX actually follows some of the same concepts that SGML does. However, TeX renders its files into DVI (Device Independent) that can then be rendered into another format. Unfortunately, DVI can't be easily converted into anything other than printer languages (PostScript, PCL), making it hard to use to generate HTML. TeX is installed or is available with most Linux distributions. TeX is available on almost all distributions as LaTeX or TeTeX. Either should work for you.

LyX

Optional − http://www.lyx.org/

LyX provides the power of writing SGML with the ease−of−use of a regular word processor. It's not a WYSIWYG program, but more WYSIWYM (What You See Is What You Mean) application, since what you see on the screen isn't necessarily what happens after the SGML processor is done with it.

Emacs (PSGML)

Optional − http://www.lysator.liu.se/~lenst/about_psgml/

Emacs has an SGML writing mode called psgml. Anyone with experience writing in this mode is welcome to e−mail the author of this document. psgml is a major mode in Emacs designed for editing SGML and XML documents. It provides "syntax highlighting" or "pretty printing" features that make SGML tag stand out, a way to insert "tags" other than typing them by hand, and the ability to validate your document while writing. For users of Emacs, it's a great way to go, and many believe it to allow more versatility than any other SGML documentation tool. It works with DocBook, LinuxDoc and other DTDs equally well.

WordPerfect 2000

Not recommended − http://www.corel.com/

WordPerfect 2000 for the MS Windows platform has limited support for SGML and DocBook 3.0. WordPerfect 2000 for Linux has no SGML capabilities.

DocBook: The Definitive Guide

Optional (but recommended) − http://www.docbook.org/

This book was released by O'Reilly in October 1999, and is a great reference to DocBook. I have not found it to be a great practical book, and much of the emphasis is on XML, but the DocBook tags for version 3.1 are all listed in a handy format. You can pick it up at the book vendor of choice. The entire book is also available online at the above URL.

Aspell

Optional − http://aspell.sourceforge.net/

This spell checking application can work around SGML tags, and only spell check the content within the tags. Default spell checkers like ispell will try to spellcheck the tags, causing errors at every new tag.

Cygnus DocBook Tools

Optional (may be Red Hat specific) − http://www.redhat.com/

Red Hat distributes three packages, possibly starting with 6.2, that include DocBook support and some tools. They provide for only rendering to HTML and PDF, but they're easily installed if you have Red Hat, allowing you to focus more on writing than wrestling with the tools. TeTex 0.9, Jade, and Jadetex must be installed first.

Getting Started with DocBook

This section covers the new method of writing LDP documentation, using the DocBook 3.1 DTD. We'll cover getting, installing, and using tools, along with an introduction to DocBook tags. Since there are over 300 DocBook tags, we won't cover them all here. Really interested readers can go to http://www.docbook.org for more information.

For New Authors

If you are a new to the LDP and want to pick up an unmaintained HOWTO or write a new HOWTO or mini−HOWTO document, contact the HOWTO coordinator at ldp−discuss@lists.linuxdoc.org. This is to make sure the HOWTO coordinator can know who is working on what documentation. Also note that all HOWTO submissions must be in SGML format using the DocBook 3.1 DTD. The mini−HOWTO submissions may be made in either SGML or HTML formats, but only SGML−formatted submissions will be included in printed versions of the HOWTOs.

Mailing Lists

There are a few mailing lists to subscribe to so you can take part in how the LDP works. First is ldp−discuss@lists.linuxdoc.org, which is the main discussion group of the LDP. To subscribe, send a message with the subject reading "subscribe" to ldp−discuss−request@lists.linuxdoc.org. To unsubscribe, send an e−mail with the subject of "unsubscribe" to ldp−discuss−request@lists.linuxdoc.org.

Downloading and installing the tools

Manual using jade/openjade

This is the quick and dirty way that should work for all distributions, no matter what distribution you're using.

1. Create a base directory to store everything such as */usr/local/sgml*. We'll call this *$_toolroot* form here on.
2. Install Jade, DocBook DTD, and DSSSL such that the base of each is under *$_toolroot* (creating *$_toolroot/jade−1.2.1*, *$_toolroot/dtd*, *$_toolroot/dssl*)
3. You'll need to set the *SGML_CATALOG_FILES* environment variable to the catalogs that you have under *$_toolroot*. You can do this with the command: *$ENV{'SGML_CATALOG_FILES'} = $_toolroot/dtd/docbook.cat;$_toolroot/dsssl/docbook/catalog;$_toolroot/jade−1.2.1/dsssl/catalog*
4. Now you can start using Jade. To create individual HTML files: *$_toolroot/jade−1.2.1/jade/jade −t sgml −i html −d $_toolroot/dsssl/docbook/html/docbook.dsl howto.sgml*
5.

To create one large HTML file, add *−V nochunks* to the jade command.

sgmltools

Unlike previous versions of sgmltools, you will require sgmltools version 2.x for use with DocBook. Since the backend programs have all changed, you'll also need to forget the sgml2xxx style of programs (sorry). Since most major distributions ship with sgml 1.x, you'll need to remove the sgml 1.x package and install either a 2.0 version, or a CVS version. To get the latest CVS source code version, you can use the following set of commands:

```
CVSROOT=:pserver:cvs@cvs.sgmltools.org:/home/cvs
export CVSROOT
cvs login
cvs -z6 get sgmltools
```

The CVS password is 'cvs'. Once downloaded, You can just use

```
./compile
make
make install
```

to install sgmltools. For Red Hat−based systems (using RPM) you can use the rpmfind command to get the latest sgmltools. The rpmfind program is available at http://www.rpmfind.net/. Make sure you get sgmltools and not sgml−tools, as the latter is sgml−tools 1.0.9. For Debian−based systems, apt−get will retrieve the right package for you:

```
# apt-get install sgmltools
```

As with the RedHat, be sure to get sgmltools and not sgml−tools.

Cygnus DocBook Tools

These tools are provided with Red Hat 6.2. Make sure the following packages are installed:

- sgml−common
- docbook
- stylesheets

Red Hat has the latest version on their web site: http://www.redhat.com/support/errata/RHBA−2000022−01.html.

Download/get/sneakernet the RPMs to your machine and install in the usual manner (become root, then rpm −Uvh *filename*). Once the RPMs are installed, you can use the following commands to render DocBook:

```
db2html filename
```

Renders DocBook into HTML. A subdirectory with the filename (minus the .sgml extension) is created and the HTML files are placed there.

```
db2pdf filename
```

Renders DocBook into a PDF file.

Writing SGML by hand

Must of this is covered by Jorge Godoy's Using DocBook document. Those interested can read it at http://metalab.unc.edu/godoy/using−docbook/using−docbook.html for writing DocBook using your favorite text editor.

Writing SGML using LyX

New documents

You can easily start a new HOWTO using LyX. Use the *File−>New from template...* menu command to bring up the template listings. Select *Templates* on the right side of the screen, then select *docbook_template.lyx* in the file listing. Select OK, and you'll have a new document. Fill in the items, such as title, abstract, and author name, then start writing.

Existing documents

If you have an already−existing LyX, TeX, or text document, you can import it into LyX with the *File−>import* command. Once the file is imported, go to *Layout−>Document...* In the popup window, under Style, select *SGML (DocBook Article)*. You'll be asked if you want to convert all text over, and say Yes. You will need to reapply most tags, but it's a fairly simple matter of selecting text and changing the style. Many LyX functions have a keyboard shortcut to assist you.

Exporting documents to SGML

Once your document is written or converted, save it in LyX format. This will allow you to edit future versions easily. Then, go to *File−>Export−>as DocBook...* and the file will be exported in DocBook.

Writing SGML using PSGML

Not written

Unfortunately, I don't use Emacs. If you use PSGML for writing/validating SGML, please e−mail the author your directions and I'll be happy to add them here as a service for other users.

Getting Started with LinuxDoc

This section shows how to get involved in writing your own LDP documentation. Getting and setting up the tools, making contact with the LDP in general, and distributing what you know to all the Linux users out there.

For New Authors

If you are a new to the LDP and want to pick up an unmaintained HOWTO or write a new HOWTO or mini−HOWTO document, contact the HOWTO coordinator at ldp−discuss@lists.linuxdoc.org. This is to make sure the HOWTO coordinator can know who is working on what documentation. Also note that all HOWTO submissions must be in SGML format (using DocBook or LinuxDoc DTD). The mini−HOWTO submissions may be made in either SGML or HTML formats, but only SGML−formatted submissions will be included in printed versions of the HOWTOs.

Mailing Lists

There are a few mailing lists to subscribe to so you can take part in how the LDP works. First is ldp−discuss@lists.linuxdoc.org, which is the main discussion group of the LDP. To subscribe, send a message with the subject reading "subscribe" to ldp−discuss−request@lists.linuxdoc.org. To unsubscribe, send an e−mail with the subject of "unsubscribe" to ldp−discuss−request@lists.linuxdoc.org.

Downloading and installing the tools

sgmltools

Download the sgmltools package from http://www.sgmltools.org/, or directly from your distribution. The source files from sgmltools.org is in source code format, so you will have to compile the source code for your machine. Using a pre−built package for your distribution is easier, as you don't have to compile it and potentially run into compilation issues (that is, if you're not a coder). With RedHat, the sgmltools is included with the distribution. If not, you can download it from ftp.redhat.com or any of its mirrors as part of the main distribution. If you're using Debian, it too has sgmltools in the standard distribution. If you don't have the package installed, you can use the apt−get command to download and install the package for you:

```
# apt−get install sgml−tools
```

For more information on the Debian package, you can look at http://www.debian.org/Packages/stable/text/sgml−tools.html If compiling from source, all you need to do is:

```
# tar −zxvf sgmltools−x.x.x.tar.gz
# cd sgmltools−x.x.x
# ./configure
# make
# make install
```

Replace sgmltools−x.x.x with the actual version of the sgmltools package you're using. The current version as of this writing that supports LinuxDoc is 1.0.9. The version that supports DocBook is 2.0.2. Both are available at the above web site. Once the tools are installed, you have a number of commands available to you.

- sgmlcheck file.sgml− Checks the syntax of a given document.
- sgml2html file.sgml− Converts an SGML file into HTML. Creates a file.html file that contains the Table Of Contents, then creates file−x.html files where x is the section number.
- sgml2rtf file.sgml− Converts an SGML file into Rich Text Format (RTF). Creates two files, the first being file.rtf that contains the TOC, and a file−0.rtf that contains all the sections.
- sgml2txt file.sgml− Converts an SGML file into ASCII text. The TOC and all sections are all put into file.txt.
- sgml2info file.sgml− Blah SGML blah INFO, used by the info command. All output is sent to file.info.
- sgml2latex file.sgml− Blah SGML blah TeX.
- sgml2lyx file.sgml− SGML yadda LyX graphical editor. This is great if you have pre−generated SGML files and want to convert them for use in LyX.

Writing SGML by hand

Much like HTML, you can write SGML by hand, once you know all the markup codes you want to use. This section will go over as many of these codes as possible, along with practical examples of each. A nice place to start would be the SGML source for this document, which is available at the web site in the Introduction. As the SGML may be processed differently depending on the file format you go to, I'll try to list some things to know about as you're writing.

Starting out

To start a new document, create a new file in your favorite ASCII editor and start with this: <!doctype linuxdoc system> This defines the document type (LinuxDoc in our case) that the SGML processor will use when it renders the file in an output format. Nothing is rendered from this tag. Next you need to enclose the rest of your work in <article> and </article> tags. This signifies the start of the content (or article, eh?). If you're familiar with HTML, this is similar to enclosing all your content with <html> and </html>.

Header information

The first part of the content should contain general information about the rest of the content. This would be similar to the first few pages of a book, where you have a title page (title of the work, author, date of publication, table of contents, and so on). The title of the content is enclosed in <title> and </title> tags. The author is specified in <author> and </author> tags. The date uses <date> and </date>. The two remaining sections are the <abstract> and </abstract> tags, which provide an executive summary of what the content is about, and the <toc> tag, which specifies the location of the table of contents. The TOC is automatically generated by the SGML processor. We'll get into sections later on. Now, how does it all look together? Taking a nice bit of SGML code (that is, what was used to create this document) you'll see:

```
60;!doctype linuxdoc system62;
60;!−− LinuxDoc file was created by LyX 1.0 (C) 1995−1999 by 60;markk62; Tue Dec 14 15:24:03 1999
60;article62;
60;title62;HOWTO HOWTO 60;/title62;
60;author62;Mark F. Komarinski 60;/author62;
60;date62;v1.1, 14 December 1999 60;/date62;
60;abstract62;List the tools, procedures, and hints to get HOWTO authors up to speed writing. 60;
60;toc62;
```

This bit of content created the main page you see when you look at this document in RTF or HTML format, listing all the information on one page.

Sections

In order to build the Table of Contents, you need to have something to build with. Sections in the case of SGML is the same as chapters in traditional publishing. You have multiple sections, and each section can have a subsection, and each of those can have a subsection and so on. Starting your document with sections is great as it lets you create an outline of the major topics you want to cover. You can then break down these major sections into gradually smaller sections, until you have a nugget of information you can write about in a few short paragraphs. In writing this document, I actually started this way. Sections are one of the few sets of SGML tags that don't require to be closed. That is, there is no </sect> tag. Nor do you have to worry about numbering. The SGML processor will handle it all when you render the SGML into something else. Sections are started with <sect> tags. A new section is started with each <sect> tag. The first section is numbered 1. Creating subsections (like 1.1) is done with the <sect1> tag. It also starts with 1. Sub subsections (1.1.1) is done with the <sect2> tag, and also starts with 1. When the SGML processor comes across the <toc> tag, it runs through the rest of the document and builds the Table Of Contents based on the number of section tags within it. Sections are numbered and listed in the TOC and then used in the rest of the document. Sub subsections (1.1.1) do not show up in the TOC, but are put in emphasized text if possible.

Normal paragraphs

Writing paragraphs of content is just like in HTML. Use a <p> tag to specify a new line, and start writing. SGML will ignore whitespace such as tabs, multiple spaces, and newlines. When SGML comes across a <p> tag, it starts a new paragraph. Proper SGML has you put in a </p> to end the paragraph.

Enhanced Text

Every now and then you need a touch of text to stand out from the others. Either to highlight code or to list a command name. The first (emphasizing text) is done with <em> and </em> tags. Typewriter text (the second example) is done with <tt> and </tt> tags.

Lists

There are two forms of doing lists under SGML. First is an enumerated list, where each item in the list is numbered (like sections) starting with 1. 1. This is the first entry in the enumerated list. 2. This is the second. 3. Third. The code for the above list looks like this: <enum> <item>This is the first entry in the enumerated list. <item>This is the second. <item>Third. </enum> The <enum> tag specifies that the following items are going to be enumerated. The other method of writing lists is itemized, where each item merely has a star, or circle, or dot, or some other method of itemizing each item.

- This is the first entry in the itemized list
- This is the second
- Third

The above code looks like this in raw SGML:

```
60;itemize62;
60;item62;This is the first entry in the itemized list
60;item62;This is the second
60;item62;Third
60;/itemize62;
```

As you can see, the <item> tag is the same for enumerated and itemized lists. A third form of lists is the description lists. This has a term being described, and the phrase that describes it. LDP The Linux Documentation Project SGML Standard Generalized Markup Language The code to create the above descriptions is: <descrip> <tag>LDP</tag>The Linux Documentation Project <tag>SGML</tag>Standard Generalized Markup Language </descrip> This isn't quite the same as itemized or enumerated lists, but you have the entire list surrounded by a tag (<descrip> and </descrip>) and each item in the line that is a word being defined is enclosed in <tag> and </tag>. The remainder of the line is taken to be the definition of the word.

Verbatim text

Sometimes you just need to print some text the way you write it. For this, you can use the <verb> and </verb> tags to enclose a paragraph in verbatim mode. Spaces, carriage returns, and other literal text (including special characters) are preserved until the </verb>. This is verbatim text.

URLs

Also in SGML is the ability to handle Universal Resource Locators (URL) of any kind. Note that this would only work when exported to HTML mode, but other formats may use them as well. A URL doesn't have an end tag, but puts its information within the <url> tag itself. Here is a URL that points to the LDP homepage: http://www.linuxdoc.org/. And here's the code to create it: <url url="http://www.linuxdoc.org/" name="http://www.linuxdoc.org/"> The url=\"{}http://www.linuxdoc.org/\"{} tells the browser where to go, while the contents of the name=\"{}http://www.linuxdoc.org/\"{} tells the browser what to print out to the screen. In this case, the two are similar, but I could create a URL tag that looks like this: <url url="http://www.linuxdoc.org/" name="LDP"> And then looks on the page like this: LDP. However, good form suggests that you duplicate the URL in the name portion. The reason for this is if you're using something like Text or RTF output, the above tag would have no meaning. you wouldn't know what URL to use.

References

While URLs are great for linking to content outside the LDP document you're working on, it's not that great for linking within the content itself. For this, you use the <label> and <ref> tags. The <label> tag creates a point in the document where you want to refer back to later on, almost like a bookmark. Creating the <label> is easy. Find the point where you want to refer back to later on, and insert the following: <label id="Introduction"> You have now created a point in the content that you can refer to later on as "Introduction". This label actually appears in this SGML work at the front of the document. When you want to refer back to that point later on (say the section called Introduction (here)), you insert the following SGML: <ref id="Introduction" name="here"> and the SGML will know to put in a link called "here" (see above) that links back to the location of the Introduction section. The other part of references is indexing. Since LDP documentation is usually published on paper as a large collection of documents, there needs to be a way of building the index at the back of the book, based on words and subjects.

Special characters

Much like HTML, you will need to escape many non−alphanumeric characters to prevent the SGML processor from interpreting them as SGML code. Here's a list of the SGML codes used. More are listed in the sgmltools User's Guide located at http://www.sgmltools.org/guide/guide.html.

- Use &amp; for the ampersand (&)
- Use &lt; for a left bracket (<)
- Use &gt; for a right bracket (>)
- Use &etago; for a left bracket with a slash (</)
- Use &dollar; for a dollar sign ($)
- Use &num; for a hash (#)

- Use &percnt; for a percent (%)
- Use &tilde; for a tilde (Ü)
- Use " and " for quotes, or use &dquot for
- Use &shy; for a soft hyphen (that is, an indication that this is a good place to break a word for horizontal justification).

CVS

The LDP is in the process of providing CVS access to authors. There are a few good reasons for using CVS:

1. CVS will keep an off−site backup of your documents. In the event that you hand over a document to another author, they can just retrieve the document from CVS and continue on. In the event you need to go back to a previous version of a document, you can retrieve it as well.
2. It's great if you have many people working on the same document. You can have CVS tell you what changes were made while you were editing your copy by another author, and integrate those changes in.
3. Keeps a log of what changes were made. These logs (and a date stamp) can be placed automatically inside the document when you use some special tags that get processed before the SGML processor.
4. Can provide for a way for a program to automatically update the LDP web site with new documentation as it's written and submitted. This is not in place yet, but is a potential goal.

If you're completely new to CVS, there are a few web pages you may want to look at which can help you out:

- http://www.sourcegear.com/CVS/Docs/blandy
- https://wroclaw.art.pl/~ser/docs/cvs.html

Getting a CVS account

First you'll need to get an account at the LDP's CVS Repository. This is pretty much the root directory that is used by CVS, with various projects (HOWTOs, mini HOWTOs, etc.) created as subdirectories of that.

You will need to create a hashed password and userid for your account. The hashed password allows you to send an encrypted password to the CVS group without them needing to know your password. You can do this with the following command, from bash (or sh):

```
$ echo put_your_password_here | perl −e "print crypt(60;62;, join '',('.', '/', 0..9, 'A'..'Z', '
```

Take the output of this command, and send it with your proposed userid to cvsadmin@cvslist.linuxdoc.org. Your unique CVSROOT will be created and you'll get an e−mail with a response. When you get your response, log into your CVSROOT and make sure everything is set up properly:

```
$ export CVSROOT=:pserver:your_userid@cvs.linuxdoc.org:/cvsroot
$ cvs −d $CVSROOT login
```

(Replace the CVSROOT with what you were sent in the response e−mail).

You will be asked for your password, and then given access to the CVS Repository in read−write mode. Once you've used cvs login once and have been given access to the system, your password is stored in .cvsroot and you will not have to use cvs login again. Just set the CVSROOT and continue on. You can get the entire linuxdoc repository with this command:

```
$ cvs get LDP
```

Or you can get the SGML source for your own document with these commands:

```
$ cvs get howto/YOUR-HOWTO.sgml
$ cvs get minihowto/YOURDOC.sgml
```

Also available is The Commit List, which is an e−mail sent for each change anywhere in the repository. Note that this is a high−volume list. You can subscribe by sending an empty e−mail to commits−subscribe@cvslist.linuxdoc.org. You can unsubscribe by sending an empty e−mail to commits−unsubscribe@cvslist.linuxdoc.org.

Other CVS repository notes

Anonymous CVS access

Anonymous CVS access (read−only) is available:

```
$ cvs -d :pserver:cvs@anoncvs.linuxdoc.org:/cvsroot login
```

As a password, use cvs. You can then get linuxdoc modules as above. Note that changes to the anoncvs site may be a half an hour behind the main site.

CVS Files via web

You can access the CVS repository via the web at http://cvsweb.linuxdoc.org/index.cgi/linuxdoc.

Graphical access to CVS

There are graphical interfaces to CVS, and you can get a list of them at http://freshmeat.net/appindex. Search for CVS.

Updating files and CVS

CVS has a special tag that you can use to automatically insert the date and version directly into the document. This is called *Id*. By including this tag in your document, you can have that automatically change each time you change the file, allowing the revision mark to increment each time.

When you're ready to upload changes to the CVS server, use the command *cvs ci −m "comment" YOUR−HOWTO.sgml*. The −m "comment" isn't necessary, but if you don't include it, you'll be brought into the editor (usually vi, or whatever your EDITOR environment variable is) and be given the chance to add a comment about the changes.

You can follow more of the CVS discussion on the ldp−discuss list. For the time being, LDP submissions should still be sent to ldp−submit.

Distributing your documentation

Before you distribute

Before you distribute your code to millions of potential readers there are a few things you should do.

First, be sure to spell−check your document. Most utilities that you would use to write SGML have plug−ins to perform a spell check. If not, there's always the aspell program.

Second, get someone to review your documentation for comments and factual correctness. The documentation that is published by the LDP needs to be as factually correct as possible, as there are millions of Linux users that may be reading it. If you're part of a larger mailing list talking about the subject, ask others from the list to help you out.

Third, create a web site where you can distribute your documentation. This isn't required, but is helpful for people to find the original location of your document.

Validate your SGML code

Using sgmltools, or really the nsgmls command, you can validate your .sgml code against the DTD to make sure there aren't any errors.

```
nsgmls −s HOWTO−HOWTO.sgml
```

If there are no issues, you'll just get your command prompt back.

Copyright and Licensing issues

In order for an LDP document to be accepted by the LDP, it must be licensed to allow for free (as in beer) distribution and publishing. As an author, you may retain the copyright and add other restrictions (for example, you must approve any translations or derivative works). A sample license is available at http://www.linuxdoc.org/COPYRIGHT.html. If you choose to use the boilerplate copyright, simply copy it into your source code under a section called "Copyright and Licenses" or similar. Also include a copyright statement of your own (since you still own it). If you are a new maintainer for an already−existing HOWTO, you must include the previous copyright statements of the previous author(s) and the dates they maintained that document.

Submission to LDP

Once your LDP document has been reviewed by a few people and you took into account their comments, you can release your document to the LDP. Send an e−mail with the SGML source code as an attachment (you may gzip it if you like) to ldp−submit@lists.linuxdoc.org.

Be sure to include the name of your HOWTO in the subject line, and use the body to outline changes you've made and attach your HOWTO. This allows the maintainers to do their jobs faster, so you don't have to wait for your HOWTO to be updated on the LDP web site. If you don't hear anything in 7 calendar days, please follow up with an e−mail to make sure things are still in process.

Style guides

This section contains notes on conventions that the LDP has agreed to in order to give all LDP documents a similar look and feel. You should keep these guides in mind when writing.

Date formats

The <date> tag in your header should be in the following format:

```
v1.0, 21 April 2000
```

FAQs about the LDP

I want to help the LDP. How can I do this?

The easiest way is to find something and document it. Also check the unmaintained HOWTOs and see if there is a subject there that you know about and can continue documenting.

I want to publish a collection of LDP documents in a book. How is the LDP content licensed?

Please see http://www.linuxdoc.org/COPYRIGHT.html. Note that this is only a guideline to authors.

I found an error in an LDP document. Can I fix it?

Contact the author of the document, or the LDP coordinator and mention the problem and how you think it needs to be fixed.