HOWTO HOWTO

Table of Contents

Table of Contents

HOWTO HOWTO

Mark F. Komarinski <markk@cgipc.com>

v1.21, 8 February 1999

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

Next Previous Contents Next Previous Contents

---

1. Introduction

1.1 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.

1.2 New versions

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

1.3 Comments

Comments on this HOWTO may be directed to the author (markk@cgipc.com).

Version History

v1.21 (February 8, 1999)

- Changed location of GNU GPL to ftp.gnu.org (per jmm@raleigh.ibm.com)
- Added CVS information. Will need to add more soon.

1.4 Copyrights and Trademarks

1.5 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.

Next Previous ContentsNextPreviousContents

2. Background on the LDP and SGML

2.1 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 some networking, 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.

2.2 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 LinuxDoc 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 different element.

Over time, the LinuxDoc DTD is going to change over to the DocBook DTD, used by others and giving the LDP a consistent look and feel to other SGML documentation. As this happens, we'll keep you updated via this HOWTO or on the mailing lists. The biggest difference between LinuxDoc and DocBook is that DocBook assigns tags to different types of content (such as commands, file names, directories, and so on) while LinuxDoc assigns tags based on the way the text should look (you can assign emphasized or typewriter for example)

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 sgmltools package 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 (DOC, PostScript, and so on). Programs like LyX (right now my LinuxDoc editor of choice) 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.

2.3 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.

sgmltools

Required

The sgmltools package contains the SGML tools needed to render SGML as any of the file formats listed above. It also contains the LinuxDoc DTD, needed to make LDP documentation. To create only SGML documentation, this is all you need. If you want to render to formats like TeX, you'll need to get those packages as well. The sgmltools package is available either with your distribution of choice, or via http://www.sgmltools.org/. Note that you will need version 1.0.9 to use LinuxDoc. Any other version is written for DocBook.

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 actually 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

The LyX program is a graphical WYSIWYM (What You See Is What You Mean) and provides a much−needed link between an easy−to−use graphical app and renderer and the sometimes−complex rules of SGML. LyX was really used to write TeX documentation, and many of the TeX rules apply in LyX. For example, while sections are automatically numbered, you can't insert whitespace (spaces and tabs) easily. It's against what TeX was designed to do. As it is, SGML often ignore the same whitespace. The LyX program can read the LinuxDoc DTD and provide a template document for you to write (or edit) your LDP documentation in a way that you're familiar with, without having to use vi and remember what the tags are for itemizing a list. LyX is available at http://www.lyx.org/.

For those of you using KDE, there is a port of LyX using the Qt libraries. You can find more information on this version at http://www.devel.lyx.org/~ettrich/klyx.html.

Emacs (PSGML)

Optional

There is an Emacs mode for writing SGML and XML documents. You can get more iformation about it at http://www.lysator.liu.se/projects/about_psgml.html.

WordPerfect 2000

Optional

The latest release of WordPerfect 2000 will have support for SGML modes. I'm not sure yet if this includes LinuxDoc, or only DocBook. If you're a beta tester of WP2k and can tell me how well it works, I'll be happy to include your notes.

NextPreviousContentsNextPreviousContents

3. Getting Started

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.

3.1 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 linux−howto@metalab.unc.edu. 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 (currently using the 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.

3.2 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.

3.3 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.

3.4 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:

```
<!doctype linuxdoc system>
<!-- LinuxDoc file was created by LyX 1.0 (C) 1995-1999 by <markk> Tue Dec 14 15:24:03 1999-->
<article>
<title>HOWTO HOWTO
</title>
<author>Mark F. Komarinski
</author>
<date>v1.1, 14 December 1999
</date>
<abstract>List the tools, procedures, and hints to get HOWTO authors up to speed writing.
</abstract>
<toc>
```

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:

```
<itemize>
<item>This is the first entry in the itemized list
<item>This is the second.
<item>Third.
</itemize>
```

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 (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 (\ {})
- 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).

3.5 Writing SGML using other tools

LyX

I'm still gushing about LyX. Okay, so I'm a bit biased towards this application because I really like it. It 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 Get 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.

To create a LinuxDoc document with LyX, download and install the application. Make sure you have TeX and sgmltools installed first (see (Installing the Tools) for more information on this). Once complete, start up LyX and select "file−>new from template..." Select "Templates" then click on `linuxdoctemplate.lyx` and you'll have a template document set up, with most of the header information that an LDP document should have. Change the data to suit your need (that is, fill in the Title, Author, Date, Abstract, and so on) and then start writing. The pull down menu in the upper left hand corner can be used to select types of content (standard, itemized and enumerated lists, sections). The exclamation point is used to emphasize text, and you can either click it and begin typing in emphasized mode, or highlight text with the mouse and click on it to emphasize the highlighted text. Many other features of SGML can be found under the Insert menu bar. You can insert URL locations, cross references, index entries, and other kinds of data. When complete with your documentation, you can save it in LyX format, then export to LinuxDoc and have the file saved with a .sgml extension. That file is then ready to be checked with sgmlcheck and rendered to the formats you want.

Emacs

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.

Other SGML tools

If there are other SGML tools out there, or even commercial ones that the LinuxDoc DTD can be used with to create LDP documentation, please let me know.

3.6 CVS basics

At this time, the LDP does not have a shared repository for you to store your content online. Hopefully this will change. 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.

So let's take a look at what you need to set up CVS on your local system to at least get started. Once network CVS access is available, many of these commands will stay the same, so don't worry about having to re−learn new tricks.

CVS init

First you'll need to create what is called a CVS repository. This is pretty much the root directory that is used by CVS, with your various projects created as subdirectories of that. It doesn't matter where this exists, though it's best if it were a dirtectory that was backed up.

For our purposes, let's say that it'll be under my home directory (/home/markk) and be called cvs.root. I can create the repository by first creating the directory `/home/markk/cvs.root`, then setting the CVSROOT environmental variable to `/home/markk/cvs.root`, then use the `cvs init` command. This creates the base repository. If you go into the `/home/markk/cvs.root` directory, you'll find a new directory called CVSROOT with basic information about the CVS setup you're using.

If you plan on using CVS for something other than plain 'ol ASCII data, you will need to change one of these files to prevent CVS from treating it as an ASCII file. This includes things like image files, .DOC files, and other binary−style data. In short, if it's not source code, and it's not HTML, TXT, or SGML, you will probably need to change the configuration. Here's what you do:

Go to another directory and check out CVSROOT. Yes, the CVSROOT directory is in CVS format too. To do this, make sure that the environmental variable CVSROOT is still set to /home/markk/cvs.root (or your CVS repositorty directory). Then use the command `cvs co CVSROOT` to "check out" the CVSROOT module. Enter the CVSROOT directory and with your favorite text editor, edit the `cvswrappers` file. There are no entries in this file, but the last line looks something like this:

```
#*.gif −k 'b'
```

If this line were not commented out, it would tell CVS that all files that end in .gif are binary files, and to not treat it as ASCII. We want this. So uncomment the line, and add lines of a similar format for other binary files that you want to cover with CVS, one per line.

Once you're done with the editing, use `cvs diff .` to make sure that cvswrappers is the only file that changed, then use `cvs commit cvswrappers` to commit the changes back into the repository. You can then delete the CVSROOT directory from the temporary directory. NEVER DIRECTLY MODIFY, DELETE, OR ADD FILES TO A REPOSITORY. ALWAYS USE THE CVS COMMAND TO DO THIS.

Creating a CVS project

Now that your repository is set to make new projects, let's create one. In order to do this, you'll need some sort of content already written. You'll also want this in a directory by itself. For example, if I wanted to create a project for the HOWTO−HOWTO, I'd have a directory setup like this:

```
wayga.auroratech.com::/home/markk/howto> ls −la
total 40
drwxr−xr−x   2 markk    console      1024 Feb  8 10:51 .
drwxr−xr−x 100 markk    users        6144 Feb  8 10:49 ..
−rw−r−−r−−   1 markk    console     32555 Feb  8 10:51 howto.sgml
wayga.auroratech.com::/home/markk/howto>
```

Again, make sure that the CVSROOT environment variable is set to `/home/markk/cvs.root`, and use the `cvs import` command to create a new project:

```
wayga.auroratech.com::/home/markk/howto> cvs import howto R1 start
N howto/howto.sgml

No conflicts created by this import

wayga.auroratech.com::/home/markk/howto>
```

What you don't see in this is when I was brought into the editor and was asked to add a comment for this file. By default this should be the contents of the EDITOR environment variable, meaning it'll probably be vi. In the cvs command, you're using the `import` command, telling CVS that you want to import the files in the current directory. The `howto` tells CVS that the name of the project is HOWTO. The `R1` and `start` reflect vendor and branches. We don't need to worry about this, since you're writing documentation. Writers of code will have to worry about it, as it has to do with code forking and the like. CVS experts are invited to e−mail the author to explain these concepts a bit more.

The file (in my case, just the howto.sgml) is added to the newly−created project. I can verify this by going to another directory and using the command cvs co howto, which will extract the howto.sgml file:

```
wayga.auroratech.com::/home/markk/baz> cvs co howto
cvs checkout: Updating howto
U howto/howto.sgml
wayga.auroratech.com::/home/markk/baz> ls −la
total 8
drwxr−xr−x   3 markk    console      1024 Feb  8 10:59 .
drwxr−xr−x 101 markk    users        6144 Feb  8 10:59 ..
drwxr−xr−x   3 markk    console      1024 Feb  8 10:59 howto
wayga.auroratech.com::/home/markk/baz>
```

I can then go into the howto directory, which now has a CVS directory within it and the howto.sgml file. I can edit that howto.sgml file as I need, then use the `cvs commit howto.sgml` command to commit changes I made back into the project. Each time I perform a commit, I have to add a comment which lists the changes I made to the document. Each commit also gets a new version number, allowing me to go back to any pervious version at any time.

Adding and removing files from a CVS project

I can add or remove files from a project if I have changes. For example, I could add in the text−formatted version of the HOWTO−HOWTO, called howto.txt. To do this, check out the current howto project into an empty directory, copy the howto.txt file into the project directory, use the`cvs add` command to add the file to the project, then use `cvs commit` to commit changes back into the main repository:

```
wayga.auroratech.com::/home/markk> mkdir baz
wayga.auroratech.com::/home/markk> cd baz
wayga.auroratech.com::/home/markk/baz> cvs co howto
cvs checkout: Updating howto
U howto/howto.sgml
wayga.auroratech.com::/home/markk/baz> cd howto
wayga.auroratech.com::/home/markk/baz/howto> cp ~/howto.txt .
wayga.auroratech.com::/home/markk/baz/howto> cvs add howto.txt
cvs add: scheduling file 'Howto.txt' for addition
cvs add: use 'cvs commit' to add this file permanently
wayga.auroratech.com::/home/markk/baz/howto> cvs diff .
cvs diff: Diffing .
cvs diff: howto.txt is a new entry, no comparison available
wayga.auroratech.com::/home/markk/baz/howto> cvs commit howto.txt
RCS file: /home/markk/foo/howto/howto.txt,v
done
Checking in howto.txt;
/home/markk/foo/howto/howto.txt,v  <−−  howto.txt
initial revision: 1.1
done
```

```
wayga.auroratech.com::/home/markk/baz/howto>
```

To remove a file, use the `cvs delete` command. You will need to check out the project, delete the file from the project directory, then use `cvs delete` and the filename, then use cvs commit to commit changes back into the main repository.

```
wayga.auroratech.com::/home/markk> mkdir baz
wayga.auroratech.com::/home/markk> cd baz
wayga.auroratech.com::/home/markk/baz> cvs co howto
cvs checkout: Updating howto
U howto/howto.sgml
U howto/howto.txt
cd hwayga.auroratech.com::/home/markk/baz> cd howto
wayga.auroratech.com::/home/markk/baz/howto> rm howto.txt
wayga.auroratech.com::/home/markk/baz/howto> cvs delete howto.txt
cvs remove: scheduling 'Howto.txt' for removal
cvs remove: use 'cvs commit' to remove this file permanently
wayga.auroratech.com::/home/markk/baz/howto> cvs commit .
cvs commit: Examining .
Removing howto.txt;
/home/markk/foo/howto/howto.txt,v  <--  howto.txt
new revision: delete; previous revision: 1.1
done
wayga.auroratech.com::/home/markk/baz/howto>
```

The file is then removed from the repository.

3.7 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 (emacs, LyX, other text editors) have plug−ins to perform a spell check. If not, there's always the ispell program, installed in just about every distribution. Also use the sgmlcheck command with sgmltools to verify you have correct SGML tags.

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.

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 in general. Send an e−mail with the SGML source code (you may gzip it if you like) to ldp−submit@lists.linuxdoc.org. 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.

NextPreviousContentsNextPreviousContents

4. Style guides

This isn't a hard and fast guide on writing good documentation (yet), but consider it a bunch of hints to help you along as you write.

- Be clear. Everyone needs to know what you're talking about.
- Use examples where possible. It lets everyone see what you're talking about.
- Organize. Don't jump between unrelated topics in the same section.

You can get many more hints from the LDP style guide located at http://www.linuxdoc.org/HOWTO/LDP−Style−Guide.html.

NextPreviousContents Next PreviousContents

5. FAQs about the LDP

5.1 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.

5.2 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.

5.3 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.

Next PreviousContents