Edición Estructurada en Emacs

Alejandro Imass

Una guía de novato a novato sobre DocBook y otros estándares SGML/XML haciendo hincapié en el uso de la edición directa y estructurada utilizando Emacs.

1. Introducción

Este artículo se centra en el uso de Emacs para editar archivos SGML y sus derivados, como por ejemplo XML y HTML. Si nunca ha usado Emacs o este programa le desagrada por alguna razón, no se preocupe, a mí también me desagradaba. Siempre me pregunté qué era lo que la gente veía en una cosa tan complicada y poco amigable como Emacs. Sin embargo, cuando empecé a escribir en Docbook comprendí inmediatamente la necesidad de una herramienta más sofisticada que un simple editor de textos a color. En un principio yo hacía mis trabajos utilizando Nedit, uno de mis editores de texto favoritos, pero no llegó a las expectativas de lo que debía ser, para mí, el perfecto editor de texto estructurado: con sangrado automático, que dejara bonitos los párrafos, que pudiera chequear la ortografía solamente de los datos y, lo más importante, que tuviera medios para la validación contextual en tiempo de ejecución de cualquier DTD. Ahora bien, ésta quizá parezca una lista de deseos muy larga y difícil de satisfacer, pero descubrí que Emacs puede hacer fácilmente todo esto y mucho, pero mucho más. Nunca advertí el poder y la flexibilidad de Emacs ya que nunca me impresionó como entorno, lo advertí en la edición de documentos SGML. De hecho, luego de descubrir Emacs gracias a la edición estructurada, este programa se ha convertido en mi navaja suiza y espero que, cuando usted termine de leer este artículo, también se convierta en la suya.

Recientemente he probado LyX con Docbook y ahora puedo concluir que este intento también tiene futuro y facilita la transición de editores WYSIWYG [1]a la manera estructurada de editar textos. Sin embargo, esta clase de cosas nunca nos darán la libertad, el poder y la flexibilidad que nos pueden dar herramientas como Emacs + PSGML, especialmente si nos encontramos en un ambiente de programación y estamos habituados a encarar los proyectos desde ese punto de vista.

Sobre el alcance de este artículo sólo diré que no se trata de una guía para SGML, Docbook o Emacs. Como ya diré más adelante, el principal objetivo es introducir y discutir el modo principal[2] PSGML de Emacs y las dificultades más importantes con las que se encontraría cualquier usuario al editar documentos SGML y XML. Este artículo trata de ser una guía para principiantes por lo que la información que se proveerá aquí debería ser suficiente para poder entender todo sin necesidad de conocimientos previos. De todos modos espero poder estimular al lector lo suficiente como para investigar aún más en el tema. Es por esto que se dan algunas referencias y enlaces a otros artículos y libros al final de este artículo.

Recordemos también que Docbook, SGML y XML se están moviendo a un ritmo muy rápido y que ya han surgido varias tendencias. Recomiendo la lectura del excelente artículo de Eric Raymond sobre Docbook en el cual se explica en qué etapa se encuentra Docbook actualmente y hacia dónde apuntan varias de las tendencias (hay un enlace al final). También deseo aclarar que este documento discute sobre el uso de SGML Docbook y no de XML Docbook, formato que parece ser una nueva tendencia. De todos modos, cualquier cosa que aparezca en este artículo probablemente se aplique también para XML Docbook que en esencia es la misma cosa. Por lo que entiendo, el modo principal PSGML es capaz de analizar sintácticamente DTD de tipos SGML y XML (o *esquemas*[3] que es el nombre que se le da en el mundo XML), lo que es otro motivo para pensar que todo lo que sea mostrado en este artículo se aplicará a ambas variantes de Docbook.

2. La Fantasía de los *Procesadores de Texto*

Yo nací en la era de los “Procesadores de textos” por lo que no tuve oportunidad de conocer lo que era componer tipográficamente hasta que descubrí UNIX y Troff hace cinco años. Aunque suene poco creíble, quedé encantado con la nueva forma de escribir usando marcas[4] y sin tener que preocuparme por el aspecto de mis documentos pero sí por el contenido y la estructura, lo cual debería ser el principal objetivo de todos los autores. Más adelante descubrí Ispell y el resto de la filosofía Unix al integrar muchas y pequeñas piezas que realmente, todas juntas, funcionan perfectamente.

Quizá, como programador, esto sea natural para mi. Me agradó el hecho de poder escribir (y mantener!) mis documentos de la misma manera en que escribo mis programas. De todos modos no quiero aburrir al lector con mis experiencias frente a una computadora por lo que, para hacer corta esta historia, un día descubrí una vieja tradición en computación llamada SGML[5], y no pude creer que las computadoras se volvieran WYSIWYG en vez de volverse WYSIWYM (un concepto extraído de la página de LyX).

He sufrido mucho y en primera persona el calvario de usar Microsoft Word™ para trabajar con documentos y manuales, en realidad no muy largos. A continuación listo algunas de mis experiencias:

- Los formatos de documentos son difíciles de integrar. Integrar el trabajo de varias personas casi siempre se convierte en una pesadilla de reformateo, terminando siempre con una docena de estilos diferentes que son difíciles de limpiar. Dar formato nuevo a un documento Ms-Word es una larga, manual y tediosa tarea.
- El número de estilos crece fuera de control y no hay manera de poner en vigor un formato. Un usuario principiante terminará con una docena de formatos distintos o simplemente los ignorará a todos. Por ejemplo, un novato en Ms-Word (o cualquier otro programa WYSIWYG) probablemente lo que hará es simplemente cambiar la fuente y el tamaño a mano, en lugar de usar los encabezados y estilos apropiados.
- En el caso de que las compañías deseen crear plantillas de documentos estándar, las mismas se impondrán muy poco y probablemente sean todas violadas o mal usadas. Es muy difícil organizar un estándar para la documentación a nivel de una corporación y probablemente las compañías terminen con una docena de diferentes “estándares” en todos los departamentos. Por ejemplo, los logotipos y otros tipos de gráficos de una corporación tienden a mutar con el tiempo en muchos tipos de formas,

proporciones y tamaños. Al final, cada usuario termina con lo que él cree que es el estándar de la compañía y, antes de que alguien se de cuenta, nadie sabrá cuál es realmente el logo oficial.

- Al trabajar con documentos de cierto tamaño, el programa tiende a volverse muy inestable, especialmente si trabajamos con imágenes u otros objetos. Un error típico de Ms-Word es la misteriosa X que por alguna razón que nadie conoce reemplaza todos nuestros gráficos. Ante este error el documento es irrecuperable y se deberá reemplazar cada gráfico a mano.
- El formato de un documento Ms-Word es binario, por lo que es básicamente inútil tratar de extraer la información dentro de ellos. Si las universidades solicitaran trabajos como tesis en estos formatos (y créanme que las universidades de aquí piden formato Ms-Word), estos serán tan útiles como papeles y desfavorecerán la creación de bases de datos de información. Por otro lado, si las universidades fueran inteligentes, pedirían los trabajos en formato SGML para así, más adelante poder crear bases de datos de conocimiento para que otros estudiantes puedan consultarlas eficientemente.
- El formato binario `.doc` parece expandirse en tamaño fuera de control cuanto más lo usemos. Solía haber una opción para compactar estos documentos pero, al parecer, ha desaparecido en las nuevas versiones de MS-Word. Esto es un real desperdicio de los recursos de una compañía dado que un usuario común desconoce esta característica: la única manera de resolver este problema es crear un nuevo documento `doc` y copiar y pegar todo el contenido del viejo documento en el nuevo.
- Los servidores de las corporaciones que usan estos formatos se alborotan y desorganizan fácilmente. Encontrar algo que usar se vuelve parecido a buscar piezas esparcidas en un campo de chatarra: se gastan horas abriendo documentos aquí y allá hasta que por fin se encuentra algo de valor.
- La reutilización de documentos es algo difícil de llevar a cabo y mantener. Por ejemplo, las presentaciones estándar de una compañía o capítulos estándar son algo muy difícil de integrar en un documento ya empezado, por lo que estas plantillas tienden a ser copiadas y modificadas por toda la compañía.
- Si el archivo binario se corrompe (lo cual sucede muy frecuentemente), no hay manera de salvar *nuestra* información, la cual está embebida en un formato propietario[6]
- La información podrá ser exportada en varios tipos de formato, pero siempre utilizando herramientas propietarias para tal tarea. No se recomienda exportar a otros tipos de formato dado que, además de la pérdida de datos que esto supone, no será una tarea trivial. Si trabajamos con MS Word y distribuimos nuestros archivos, estamos obligando a estas personas a utilizar software propietario tanto para leer como para convertir nuestros documentos a otros formatos.

De cualquier forma, podría seguir escribiendo páginas de todos los problemas que herramientas del tipo Lo Que Ves Es Lo Que Obtienes (WYSIWYG) han causado a compañías y a la gente durante años. Personalmente, tuve la experiencia de tener una secretaria que se tomaba una hora o más para escribir una simple carta (mientras buscaba el logo correcto, chequeaba virus, imprimía y tenía problemas con la red, etc), una tarea que hace diez años se realizaba en tres minutos en una máquina de escribir común que no usaba electricidad. He visto estudiantes que derrochan varios días en tratar de integrar las diferentes partes de un trabajo y a veces pierden meses de tiempo debido a la corrupción de sus documentos o por virus en los mismos. En cuanto a mí, he perdido días y semanas de trabajo en la edición de manuales largos o especificaciones de algún sistema y estoy seguro que cualquier lector al leer estas líneas se acuerda de algún caso parecido que le ha tocado vivir personalmente.

En conclusión, las herramientas para “procesamiento de textos” del tipo WYSIWYG son excelentes para

trabajos pequeños de oficina o para uso simple personal. En algunos aspectos no se les puede negar su poder y pueden encargarse de trabajos de peso mediano sin problemas. No fueron hechas para la edición profesional a gran escala y definitivamente no son recomendables para uso a nivel corporativo, en el mismo sentido en que las hojas de cálculo nunca fueron diseñadas para almacenar gran cantidad de datos de alguna corporación, una mala costumbre muy común en las empresas modernas.

Aunque las herramientas WYSIWYG son relativamente fáciles de usar por un principiante, cuando queremos implementar su uso en forma profesional la curva de entrenamiento se hace cada vez más inclinada y los estándares no pueden ser puestos en práctica con dichas herramientas.

Cualquiera que haya tenido que darles un uso serio a estas herramientas estará de acuerdo con que hay algo mal en ellas y que de seguro debe existir una forma alternativa de realizar el trabajo ¿ Alguien no está de acuerdo con que crear la Enciclopedia Británica es prácticamente imposible con herramientas como MS Word o Adobe Page-Maker ? Cuando uno mira hacia atrás en la historia de las computadoras se da cuenta de que esas alternativas siempre estuvieron disponibles, pero por alguna razón nunca fueron populares entre los usuarios mortales de computadoras[7]

3. El Mundo del Formateo de Textos

Usaré el concepto “Formateo de textos” discriminándolo de “Procesamiento de textos”[8] tal como aprendí leyendo mi primer libro de UNIX. Fue en ese libro donde por primera vez descubrí lo que era Troff y la filosofía que se esconde detrás del concepto de formateo de textos (vs. WYSIWYG).

El formateo de textos diferencia los siguientes elementos en un documento:

- Contenido
- Estructura
- Estilo

Básicamente, la filosofía se resume en que los autores deben concentrarse en el contenido y estructura antes que en el estilo de sus documentos. Dado que la mayoría de las veces la estructura está predefinida (o estandarizada en el caso de Docbook) para satisfacer el cometido del autor, éste termina concentrándose estrictamente en el contenido, lo cual debería ser su interés primario.

En el formateo de textos los autores pueden usar las más simples herramientas, como un editor de textos, o pueden utilizar editores de textos estructurados y especializados, como LyX o Emacs. En cualquiera de los casos, su información siempre es almacenada en el más natural de los formatos de computadora: *texto plano*. Esto adquiere mucho sentido dadas todas las razones que se dieron en contra de los formatos binarios en Sección 2

3.1. ¿Por qué SGML?

Resumen corto de la historia

Opuestamente a lo que la mayoría cree, SGML (el padre de XML) es un lenguaje muy antiguo, hasta tal punto que algunos lo llaman “la venganza de los cuarentones”. Este significado se debe a que hace algo más de 20 años hubo muchos entusiastas de computación que hicieron mucha fuerza en la comunidad para la implementación de estándares abiertos como SGML. Éstos fueron ignorados por el mercado o quizá anulados del mismo por intereses comerciales. De cualquier modo, SGML se ha usado en compañías e industrias muy importantes durante muchos años. De hecho, muchas corporaciones importantes han almacenado su experiencia y conocimiento corporacional en SGML desde hace ya mucho tiempo[9].

XML es simplemente un subconjunto simplificado de SGML lo que significa que cualquier documento XML es un SGML válido, pero no cumpliéndose necesariamente lo recíproco. El cometido básico de XML en nuestros días es superar lo que HTML[10] ha sembrado tras la explosión de la web en los 90. Historias sobre estos lenguajes pueden ser encontradas por toda la web, por lo que no daré más detalles sobre este tema. De todos modos, en el resto del documento me estaré refiriendo a SGML, pero en este punto el lector probablemente se de cuenta de que XML es casi la misma cosa, sólo que más simple. Cuando hablamos de transformaciones a otros formatos, existen algunas distinciones importantes entre SGML y XML. SGML usa las hojas de estilo DSSSL las cuales son escritas en Scheme, un dialecto de Lisp. XML ha desarrollado un estándar más simple de transformación llamado XSLT el cual se escribe en XML lo cual hace que sea mucho más fácil de escribir y mantener. Dado que todo esto no es un punto fundamental de este artículo, recomiendo a los lectores interesados en el tema que lean el excelente artículo de Eric Raymond:

http://www.tldp.org/HOWTO/Docbook-Demystification-HOWTO/
(http://www.tldp.org/HOWTO/Docbook-Demystification-HOWTO/)

Finalmente y para aclarar la pregunta implícita en el título de esta sección, SGML (o XML) es una buena idea principalmente porque es un lenguaje extensible que puede ser fácilmente transformado en otros estándares cuando estos salgan al público. Esto significa que si usted escribe o mantiene sus documentos en SGML, se asegurará que sobrevivan a cualquier cambio en la tecnología o en los estándares y puedan ser transformados a cualquier otro tipo de estándar que eventualmente surja. Más aun, SGML es el antiguo y estable ISO 8879:1986.

3.2. Breve vistazo a SGML

La denominación SGML viene de *Standard Generalized Mark-up Language* (algo así como Lenguaje Estándar Generalizado de Etiquetas). A su vez, XML viene de eXtensible Mark-up Language (Lenguaje Extensible de Etiquetas) y es simplemente un subconjunto simplificado de SGML (no una implementación, ver sección anterior). Es un lenguaje de computadoras, pero no es un lenguaje

procedural (o sea, las instrucciones no son ejecutadas en algún orden específico), sino un lenguaje declarativo.

Los lenguajes de etiquetas (Mark-up) se basan justamente en el uso de <la etiqueta> (tag). Ésta define la estructura de un modo muy parecido a lo que sucede en las bases de datos. Todo lo demás es contenido o, para utilizar la jerga SGML, caracteres de datos (*CDATA*). Desde el punto de vista del estándar, SGML sólo define reglas básicas sobre el uso y anidación de las etiquetas. Por ejemplo, se encuentra explícito en el lenguaje que cada etiqueta debe tener una etiqueta de final (end tag) correspondiente y que la anidación de etiquetas debe ser perfecta, lo que significa que las etiquetas internas deben ser cerradas antes de las etiquetas externas. Por supuesto, las definiciones SGML son mucho más complejas que esto, pero básicamente esas son las reglas principales. XML también tiene reglas estrictas como éstas. Hay etiquetas excepcionales las cuales no tienen una etiqueta de final correspondiente y son del formato <tag/>. Además, a las etiquetas también podemos pasarles parámetros como es el caso de la etiqueta `<BODY>` en HTML, en la cual se puede, por ejemplo, especificar el color de fondo de la página con el parámetro `BGCOLOR="WHITE".`.

Dadas estas reglas básicas, todo lo demás se deja a discreción del usuario. Esto significa que el nombre de las etiquetas, sus parámetros y las implementaciones específicas de las reglas de anidación pueden ser realizadas por la persona que escribe en SGML. Ésta es la belleza verdadera del lenguaje ya que permite que cualquiera invente su propio estándar de etiquetas. Si lo pensamos detenidamente entonces SGML es en realidad un lenguaje para describir estructuras flexibles de datos que pueden ser fácilmente manipuladas y transformadas por herramientas, llamadas analizadores sintácticos (parsers). Las estructuras de datos SGML son, por mucho, más flexibles que las tablas RDBMS (aka SQL) para almacenar cosas complicadas como libros, pero no sólo se limitan a manipular documentación. De hecho, SGML y ahora también XML, están siendo utilizados para el almacenamiento en bases de datos de propósitos generales, desplazando varias implementaciones heredadas de DBMS de Relacionamiento de Objetos (Object-Relational DBMS).

Ahora bien, si cualquiera puede realizar su propio lenguaje de etiquetas con SGML, tiene que haber un modo de poner en vigor las reglas que hemos inventado. Esto es implementado por lo que se denomina como DTD (Document Type Definition, o en español Definición del Tipo de Documento). Un DTD define los nombres de las etiquetas, el orden legal de las ocurrencias y las reglas de anidación para una implementación SGML particular. A la vez, los DTDs se escriben en SGML y se representan como un simple archivo en el sistema. En especial, Docbook es un DTD que se usa para definir *libros* y *artículos*.

El uso de DTDs impone ciertas condiciones. Primero sobre la persona que va a utilizarlas, o sea, la persona deberá tener algún conocimiento básico de la DTD antes de empezar a escribir en ese formato. También es conveniente usar un editor estructurado que tenga la capacidad contextual de validación. Estos editores son muy útiles ya que nos irán indicando las etiquetas que están permitidas en cierto punto de nuestro documento (irán analizando las reglas de la DTD por nosotros). Aún más, podremos usar un analizador sintáctico que valide los documentos usando un cierta DTD de nuestro sistema y que indique todas las inconsistencias.

Bueno, esto es básicamente todo lo que se necesita saber para empezar a trabajar con SGML. Todos estos conceptos se clarificarán a medida que avancemos la lectura, especialmente en la siguiente sección que

discute cómo crear un documento SGML usando Emacs.

4. SGML usando Emacs

La palabra Emacs podría ahuyentar a mucha gente que lea este artículo. Yo los entiendo. No puedo entender por qué diablos un editor tiene que ser tan complicado y obscuro. Es por esto que nunca me esforcé mucho por aprender a usarlo. Soy tan sólo un hombre simple y haragán y amo a los editores simples y poderosos como Nedit, del cual pienso todavía que es un gran editor. Sin embargo, y créanme, cuando descubran la edición SGML utilizando Emacs (lo cual, con suerte aprenderán aquí), realmente se sorprenderán. De hecho, estoy seguro de que quedarán tan impresionados que nunca más querrán abandonar Emacs para ninguna de sus necesidades de edición. Están leyendo estas palabras de alguien que solía evitar Emacs a toda costa y ahora lo usa para todas las cosas que se puedan imaginar.

Aprender Emacs es un viaje largo, pero no tiene por qué ser aburrido. Se debe entender que este programa no es tanto un editor de textos, sino todo un ambiente. Luego, si se lo lleva poco a poco, terminaremos por acostumbrarnos a él y, eventualmente amarlo.

Antes de empezar con la parte práctica de esta guía se necesitará instalar todas las siguientes cosas para interpretar los ejemplos.

4.1. Requisitos del Sistema

Necesitará verificar su sistema y asegurarse de que cuenta con todas las herramientas descritas abajo instaladas y funcionando.

- Alguna clase de sistema operativo UNIX o tipo-UNIX. Lo siento, este artículo no es para usuarios de Windoze. No estoy discriminando injustamente, es sólo que no tengo ni idea de cómo funcionan las herramientas en este ambiente y tampoco tengo intenciones de averiguarlo. Yo por ejemplo, uso el Sistema Debian con el Núcleo Linux. [11].
- Se necesitará alguna versión de GNU Emacs. Probablemente todo esto también funcione con XEmacs pero francamente no sé cuáles son las diferencias entre ambos paquetes. De todos modos, si usted es un usuario de XEmacs probablemente sepa qué hacer para conseguir que todo esto camine. Yo uso Emacs 21 el cual viene con unos coquetos menús y una barra de herramientas en X. Esta última versión probablemente sea la más adecuada para los principiantes.
- PSGML: (Tomado de la documentación de PSGML) “Es un modo principal para editar documentos SGML y XML. Funciona con GNU Emacs 19.34, 20.3, o mayores o con XEmacs 19.9. PSGML contiene un simple analizador sintáctico SGML y puede trabajar con cualquier DTD. Entre las funciones que provee se destacan menús y comandos para insertar las etiquetas que sean contextualmente válidas, identificación de errores estructurales, edición de los valores de los atributos en una ventana aparte con información sobre tipos estándar y una edición estructurada. ”

La discusión sobre este paquete y por qué Emacs puede ser tan poderoso en la edición estructurada como SGML o HTML será el principal objetivo de este artículo. De hecho, el modo HTML de Emacs es muy poderoso y hace que la creación y edición de páginas web sea una pasada. Doy ánimos a los usuarios a que, después de leer este artículo prueben este modo principal de Emacs. Algunas personas son perezosas y prefieren los editores HTML de tipo visual. Si bien son muy fáciles de usar, no son prácticos a la hora de crear páginas web dinámicas. El código que generan casi siempre es muy sucio (en particular el código generado por MS-FrontPage es horrible) o lleno de datos inútiles. Esto hace que se haga difícil el cortar y pegar piezas aquí y allá para crear páginas dinámicas usando, por ejemplo Perl/CGI o PHP. Además, si nos consideramos a nosotros mismos programadores y admitimos la necesidad de una de estas herramientas visuales, probablemente nos encontremos en el sitio equivocado y deberíamos considerar cambiar nuestra carrera al diseño gráfico.

- SGML-Base y SGML-Data: Así es como son llamados en Debian. Se deberá contar con la instalación equivalente en la distribución usada. De cualquier modo, esto debería instalar el archivo de catálogo de SGML base y los DTDs básicos (se explicará todo esto más adelante). Estoy casi seguro de que esto es lo que deja que nuestros documentos usen los identificadores públicos en lugar de los identificadores del sistema.

  En cualquier documento SGML, la primera línea debe ser una declaración SGML o una declaración DOCTYPE de la forma:

  ```
  <!DOCTYPE article PUBLIC "-//OASIS//DTD Docbook V3.1//EN" >
  ```

  Esta línea le dirá al analizador sintáctico frente a qué tipo de documento se encuentra y contra qué DTD debe validar. En el ejemplo de arriba, el documento es un “artículo” definido en el DTD Docbook. Si todo está instalado correctamente, el sistema debería ser capaz de encontrar el DTD sin que le tengamos que especificar caminos a los archivos. Si el encabezado de arriba no llegara a funcionar, todavía tendremos la posibilidad de agregar lo que se llama un “identificador de sistema” para que así el analizador sintáctico pueda encontrar el DTD. Los analizadores sintácticos toman la siguiente forma:

  ```
  <!DOCTYPE book PUBLIC "-//OASIS//DTD Docbook V4.1//EN"
      "/usr/lib/sgml/dtd/docbook-3.1/docbook.dtd">
  ```

  Estoy bastante seguro de que los archivos de catalogos SGML que se piden como requisitos es lo que deja al usuario usar solamente identificadores públicos, lo que hace que los documentos sean mucho más portables. Si algún lector puede contribuir con algún comentario adicional en esta parte, agradecería la retroalimentación.

- SGML-Tools, SGML-Tools Lite, Docbook-Tools, Debian-Doc o Linux Doc Tools. Esto depende de la distribución Linux usada y las tareas que se quieran realizar. Estos paquetes se denominan de forma

diferente en cada distro pero realizan las mismas tareas. Se encargan de instalar guiones que facilitan la conversión de SGML a cualquiera de los formatos HTML, PS, PDF, RTF, TeX, etc. Estos guiones probablemente causen dependencias a otros paquetes como Jade y TeX. También necesitaremos instalar el DTD de Docbook, las hojas de estilo DSSSL de Docbook, XML y XSL (si eventualmente queremos trabajar con XML-Docbook) y el paquete Docbook-Utilities. Después de que hayamos instalado todas estas utilidades seremos capaces de ejecutar órdenes como **db2ps**, **db2html**, **db2rtf** y demás.

He aquí algunas de las dependencias que puedo deducir de la información provista por los paquetes en Debian:

- La conversión a texto requiere groff-base
- La conversión a LaTeX requiere tetex-base, tetex-bin y tetex-extra.
- La conversión a Info requiere Jade.
- La conversión a PostScript requiere TeX, DVI, GhostScript y otras herramientas PS. A su vez, si queremos ver los archivos `.ps` deberemos tener instalado GhostView u otro programa equivalente.

La lista de arriba debe tomarse como algo representativo y variará según el sistema (si hablamos de Linux dependerá en la distribución y versión). De todos modos, dado que SGML y Docbook son herramientas bastante estándar en los sistemas operativos libres, es muy común que su sistema tenga un macro paquete que instale todo lo que necesitemos con relativa facilidad.

4.2. Hola Mundo

A este punto el lector debería ser capaz de iniciar Emacs invocando **emacs** en el intérprete de órdenes o seleccionando la opción en nuestro escritorio. Notar que Emacs tendrá el mismo comportamiento tanto en X Window como en la consola de caracteres. Creo que la única diferencia es que en el modo de caracteres los menús deben ser activados con el teclado, en lugar de utilizar el ratón. En realidad no puedo explicar mucho sobre este punto ya que nunca he usado Emacs en modo de consola puro.

Una vez que Emacs se haya ejecutado nos encontraremos en el búfer *scratch*.[12] Podremos crear un nuevo archivo usando los comandos **C-x** C-x C-f

Nótese que **C-x** hace referencia a presionar la tecla **Ctrl** y luego la tecla **x**. La secuencia **M-x** hace referencia a presionar la tecla **Alt** y luego la **x**. En Emacs, la tecla **Alt** puede emularse presionando (y soltando) primero la tecla **Esc** siguiendo la tecla que corresponde. De hecho, en algunos sistemas la tecla **Alt** no funcionará y nos veremos obligados a usar la tecla **Esc**.

Emacs preguntará por el archivo en la parte inferior de la pantalla, la cual se llama mini-búfer. Si se escribe el nombre de un archivo que no existe en el directorio actual (casi siempre ~/), Emacs creará un nuevo archivo. Para empezar, escríbase `hello.sgml` y presiónese **enter**.

> Para la gente nueva en Emacs, este trajeteo entre cuadros puede resultar un poco confuso. Aunque, Emacs automatiza la mayoría de estas conmutaciones, si en algún punto nos encontramos perdidos, para volver a empezar podemos hacer click en el mini-búfer y presionar **C-g** algunas veces. Esta orden le dice a Emacs que aborte las órdenes previas.
>
> Si por alguna razón nuestra pantalla está dividida en más de un cuadro, podremos "maximizar" (para usar un término familiar) cualquiera de estos cuadros simplemente haciendo click en él y luego presionando **C-x 1**. Además, podremos dividir la pantalla de muchas maneras diferentes. Pruébese esto presionando **C-x 2**, **C-x 3**, y demás combinaciones. Estando en X-Window es realmente una buena idea tener la pantalla separada en cuadros, ya que podremos seleccionar cada uno de ellos por separado con un simple click de ratón.

Si se ha instalado correctamente PSGML se debería ver `Loading psgml...done` en el mini-búfer. También, en la barra de estado (la línea que está justo arriba del mini-búfer), se debería ver `SGML` indicando que se ha activado el modo principal SGML.

> Emacs trabaja en los modos principal y secundario. Sólo puede haber un único modo principal activado a la vez. El modo SGML en el modo principal en el cual estaremos trabajando nosotros. Los modos secundarios se caracterizan por realizar tareas específicas y se puede tener varios de ellos ejecutándose dentro de un modo primario. Por ejemplo, existen varios modos secundarios útiles como auto-fill-mode e ispell-mode, los cuales nos ayudarán mucho cuando trabajemos con archivos SGML. El modo secundario Ispell verificará nuestra ortografía en tiempo real, mientras que el modo secundario auto-fill nos ayudará a mantener nuestros párrafos justificados correctamente.

He aquí el archivo completo para un documento tipo "Hola mundo":

Ejemplo 1. Ejemplo de código para Hola Mundo

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD Docbook V3.1//EN" []>
<book>
  <title>Mi primer libro</title>
  <chapter>
    <title>Mi primer capítulo</title>
    <sect1>
      <title>Mi primer sección</title>
      <para>
        Mi primer párrafo
     </para>
```

```
    </sect1>
  </chapter>
</book>
```

Se podría escribir este listado carácter a carácter en cualquier otro editor (excepto quizá con LyX), o se puede sacar provecho de las ventajas que tiene el modo principal PSGML de Emacs para la edición estructurada. Aquí es donde comienza la parte interesante...

4.3. Primer Paso: La declaración del Documento

La primera cosa que siempre se deberá hacer al editar un documento Docbook es escribir la Declaración del Tipo de Documento, y esto debe ser realizado manualmente.

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD Docbook V3.1//EN" []>
```

Esta primera línea en el documento le dirá a Emacs dónde encontrar la DTD para que pueda analizarlo sintácticamente cuando sea necesario. La DTD contiene todas las reglas para Docbook y permitirá a PSGML deducir qué cosas están permitidas, en qué lugares y cómo realizar el sangrado del documento.

4.4. Empezar a usar PSGML

Ahora que se tiene la línea DOCTYPE, muévase el cursor justo debajo y presiónese **C-c C-e**. Si nos tardamos lo suficiente en el mini-búfer se podrá apreciar que Emacs ha elegido *book* como el elemento válido. Esto es así porque en ese punto del documento no hay otro elemento válido que podamos ingresar. Ahora, presiónese enter y veamos que ocurre. Si todo funciona bien, obtendremos el siguiente código:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD Docbook V3.1//EN" []>
<book>
</book>
```

Ahora presiónese enter una vez más y luego **C-c C-e**. Obsérvese ahora que Emacs no es capaz de decidir qué etiqueta usar en este punto dado que existe más de una opción. Igual que en una shell se podrá presionar **Tab** para autocompletar pero, dado que existen muchas opciones se verá que la pantalla se divide mostrándonos una lista de las etiquetas válidas en ese punto del documento.

Si se desea salir de dicha selección, simplemente presiónese **C-g** o escríbase las primeras letras de la etiqueta (por ejemplo, escríbase “ti”), y luego **Tab** y luego **Enter**. Lo que hemos hecho es crear una etiqueta para insertar el título del documento y Emacs ya ha ubicado el cursor justo donde los necesitamos! Escríbase el título y muévase el cursor al final de la línea (como en cualquier editor, se podrá presionar **End** con tal motivo), presiónese **Enter** y luego **C-c C-e**. Escríbase ahora “ch” y luego presiónese **Tab**. ¡Acabamos de crear nuestro primer capítulo!. Sígase igual, hasta haber editado un documento parecido a `hola-mundo.sgml` listado en Ejemplo 1>.

Hasta el momento, se podrá empezar a apreciar lo que se quiso decir con un editor de textos estructurado. Y esto es sólo el comienzo, ya que la funcionalidad de PSGML se conecta con muchas características interesantes de Emacs, lo que realmente confirma que ambas cosas son las herramientas definitivas para el desarrollo de documentos SGML. Por supuesto, puede pasar que a este punto a uno todavía no le agrade tanto Emacs porque, a pesar de las ventajas vistas, todavía sigue siendo muy raro y obtuso. Así que, hagamos algunas mejoras....

4.5. Gran introducción, pero Emacs sigue sin gustarme demasiado...

Bueno, probablemente para muchos usuarios como yo, que venimos de editores de texto amigables y coquetos, Emacs sea insípido y complicado por lo que en esta sección se mostrará cómo hacer que funcione como un editor de textos normal. Se podrá ver que con un pequeño retoque Emacs puede ser tan o más amigable que el editor de textos que solíamos usar.

4.5.1. Colores de las Fuentes

La principal cosa que perdí al migrar a Emacs fue el Coloreado Sintáctico[13] y me costó mucho trabajo enterarme de cómo activarlo. En Emacs, la opción para esto se llama *Global Font Lock*. El porqué de tal nombre es un misterio para mí, pero he aquí lo que hay que hacer para activarla:

```
M-x global-font-lock
```

Obviamente, no se deberá escribir todo esto. Simplemente *global-f* y **Tab**.

4.5.2. ¡Quiero ver lo que estoy seleccionando!

Otra característica estándar molesta de Emacs es la imposibilidad de ver lo que se está seleccionando. Emacs trae el modo mark desactivado por defecto, así que lo deberemos activar con:

```
M-x transient-mark-mode
```

4.5.3. ¡Dios mío! ¿Cómo hago para desactivar el estúpido ajuste de línea?

En lo personal, ésta fue la cosa más molesta de Emacs. Es decir, hay muchas situaciones en las que, le guste o no al estándar de codificación GNU, hay que ir más allá del margen derecho. Bueno, ellos han hecho bastante difícil el desactivar esta opción estúpida, pero he aquí lo que hay que hacer:

```
M-x hscroll-mode
```

Sin embargo, esto no termina aquí, y lo siguiente no será tan trivial. Se deberá cambiar al búfer *scratch* (lo cual se podrá hacer seleccionando el menú Buffers y luego click en *scratch*). Éste es un búfer de evaluación de expresiones Lisp (lo que quiere decir que simplemente, el modo principal por defecto de este búfer es para evaluaciones Lisp, lo cual se puede ver en la línea de estado). Ahora, escríbase la línea

```
(setq-default truncate-lines 1)
```

seguida de **C-j**. Esto ejecutará la línea de código Lisp.

4.5.4. Guardando nuestras personalizaciones.

Lo que se debe hacer es crear un archivo llamado `.emacs` en nuestro directorio `home`. Éste será ejecutado cada vez que Emacs se inicie. He aquí cómo luce mi archivo `.emacs`:

```
(global-font-lock-mode)
(transient-mark-mode 1)
(hscroll-mode)
(setq-default truncate-lines 1)
```

5. Más allá de Hola Mundo

Hasta aquí hemos visto las características básicas de la edición estructurada usando Emacs. Sin embargo, hay mucho, pero mucho más que ver. He aquí algunos bonitos trucos y sugerencias que van a hacer nuestra vida como autores de documentación mucho más fácil.

5.1. Dividir nuestros documentos en diferentes archivos.

Una de las principales ventajas del Formateo de Textos sobre la Edición de Textos, es la habilidad de trabajar en archivos separados y la flexibilidad a la hora de organizar estos archivos de manera personal. Más aún, con la técnica de Formatear Textos se puede ser tan flexible con nuestra documentación, como lo somos a la hora de organizar nuestros archivos fuentes cuando programamos.

Las Entidades deben ser declaradas en la declaración DOCTYPE al principio del documento. Una declaración típica es como la siguiente:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD Docbook V4.1//EN" [
<!ENTITY presDeLaCompania SYSTEM "presDeLaCompania.sgml"> ]>
```

Como se puede deducir del ejemplo anterior, las entidades son declaradas dentro de los paréntesis rectos []. Además, puede verse que existe un identificador de sistema para la entidad, de manera que el analizador sintáctico pueda encontrar el archivo cuando procese el documento principal. El nombre de la entidad, además de identificarla dentro del documento, puede tener cualquier nombre que deseemos. Personalmente, en el ejemplo, traté de mantener el mismo nombre entre el de la entidad y el identificador del sistema, pero esto no es un requisito.

Para usar nuestra entidad, sólo deberemos usar el signo & y su nombre, por ejemplo:

```
&presDeLaCompania;
```

> **Importante:** Cuando trabajamos con entidades, no se puede incluir una declaración DOCTYPE en la misma. Esto hace que a Emacs le sea imposible saber cómo validar el documento. He encontrado una alternativa para superar esto y es explicada abajo.

5.2. Validar nuestros documentos

Como se verá al correr de esta sección, Emacs/PSGML no sólo nos ayudará a editar documentos SGML, sino que también nos ayudará a validar documentos en cualquier momento que nos encontremos editándolos. Esta es una característica estándar pura de Emacs (no de PSGML), gracias a la cual aprenderemos a valorar la edición estructurada en dicho editor de textos.

5.2.1. Validar un documento simple

En cualquier punto del documento, simplemente presionando **C-c C-v** se podrá ver la orden *nsgmls* en el mini-búfer. Presionando enter, podremos validar nuestro documento actual. Tanto si ocurre un error en la validación, como si salió todo bien, seremos notificados en una división de la pantalla, por debajo del mini-búfer.

Algo útil con respecto a esto es saber que, si hacemos click en uno de los errores y luego enter, nos veremos desplazados justo a la línea donde se encontró el error. Es más, si el error está en otros archivo que el actual, Emacs lo abrirá automáticamente y ¡nos hará saltar a donde supuestamente se encuentra el error! Ésta es una funcionalidad estándar de Emacs en integración con algunas otras herramientas de nuestro sistema. Es el mismo caso que **gdb**, **make** o un compilador java. Así de simple.

5.2.2. Validar partes (entidades) de un documento

El principal problema con la división del documento en partes separadas en archivos, o *entidades* que es como se las denomina en SGML, es que no podemos tener una declaración DOCTYPE en cualquiera de

estas partes. Si lo hiciéramos así, Emacs no tendría forma de saber dónde buscar la DTD. He aquí lo que yo hago para solucionar este problema:

1. Ir al documento principal, (aquel que tiene la declaración DOCTYPE) e ir al menú DTD -> Save Parsed DTD. Dar aquí un nombre simple como `docbook.ced` o `main.ced`. Sería bueno almacenar este archivo en el mismo directorio en el que almacenamos nuestro documento principal.
2. Al abrir la parte del documento, obtendremos el error `External entity XXXXXX not found`, donde `XXXXXX` es el nombre de la primera etiqueta que se encontró. Ignórese esto y presiónese **C-x 1** para maximizar el cuadro en el que trabajaremos. Selecciónese el menú DTD->Load Parsed DTD y sustitúyase `filename.ced` por `main.ced` (o por el nombre que le hallamos dado al archivo en 1).

Y ¡eso es todo! Ahora podremos trabajar con dicha parte del documento en Emacs de igual forma que como lo veníamos haciendo con nuestro documento principal.

5.3. Manteniendo nuestros documentos bonitos

En esta sección se hablará sobre el correcto sangrado y justificación de nuestros Docbook. Para esto usaremos algunos modos principales de Emacs y el propio modo principal PSGML.

5.3.1. La Tecla Tab

A diferencia de cualquier otro editor que yo haya visto antes, la tecla **Tab** se comporta de un modo sorprendentemente inteligente en Emacs. Por ejemplo, si nos encontramos en un modo principal distinto al modo de texto puro, al presionar la tecla **Tab** en cualquier parte del documento, esto hará que la línea en la cual estamos posicionados sangre correctamente, mientras que en cualquier otro editor de textos lo anterior quebrará la línea en dos.

5.3.2. Sangrar una región

Como modo de ejemplo, supóngase que en Ejemplo 1 hemos decidido que nuestra Sect2 sea ahora una Sect1 y queremos hacerla sangrar apropiadamente. He aquí como hacer tal cosa:

1. Selecciónese la región (**C-space** activa el bloque de la selección) y reemplácense las etiquetas. Si estamos sangrando para afuera una región deberíamos empezar a buscar y reemplazar empezando por las etiquetas exteriores. Si, por el contrario, estamos sangrando hacia adentro, deberíamos hacer lo contrario.
2. Una vez que las etiquetas tengan los nombres correctos, Emacs debería ser capaz de realizar el sangrado de la región automáticamente. Para ello, simplemente selecciónese la región nuevamente y presiónese **M-x indent-region** y Voilà!

5.3.3. Otros detalles sobre el sangrado

En esta sección se discutirán aspectos que involucran cuestiones de estilo y claridad en el código. Personalmente, en algunos casos prefiero que las líneas de *CDATA* (los caracteres de datos) estén entre las líneas que ocupan las etiquetas, y no que estas últimas estén al principio y al final de CDATA, en las mismas líneas. Los párrafos son ejemplos perfectos de esto:

```
<para>
  En esta sección se discutirán aspectos que involucran
  cuestiones de estilo y claridad en el código. Personalmente,
  en algunos casos prefiero que las líneas de
  CDATA (los caracteres de datos) estén
  entre las líneas que ocupan las etiquetas, y no que estas
  últimas estén al principio y al final de CDATA, en las
  mismas líneas. Los párrafos son ejemplos perfectos de esto:
</para>
```

versus

```
<para>En esta sección se discutirán aspectos que involucran
cuestiones de estilo y claridad en el código. Personalmente,
en algunos casos prefiero que las líneas de
CDATA (los caracteres de datos) estén
entre las líneas que ocupan las etiquetas, y no que estas
últimas estén al principio y al final de CDATA, en las
mismas líneas. Los párrafos son ejemplos perfectos de
esto. </para>
```

Téngase presente que para el sangrado de una región, Emacs seguirá el sangrado de la primera línea y a partir de este, alineará todas las demás. Luego, deberemos tener correctamente hecho el sangrado de las líneas superiores para que en las líneas inferiores sea correcto.

5.4. Ortografía

Dado que Emacs se integra pacíficamente con el sistema operativo y sus herramientas, también lo hace con Ispell y spell. Esta integración puede ser *ad-hoc* activando el modo principal Ispell o puede ser ejecutado en un modo secundario. Esta última manera de funcionar es análoga al subrayado rojo que se presenta en las faltas ortográficas de los documentos en Ms Word. He aquí algunas pistas:

- Para revisar ortográficamente un documento, presiónese **M-x ispell**.
- Para cambiar el diccionario presiónese **M-x ispell-change-dictionary**. (Pueden verse los diccionarios disponibles de Ispell en `/usr/lib/ispell`.
- Para activar el modo secundario de Ispell, simplemente presiónese **M-x flyspell-mode**.
- Si se tiene activado el modo secundario Ispell y se escucha un beep, esto significa que se ha cometido un error ortográfico en alguna palabra. Para corregirlo, deténgase la edición inmediatamente y

presiónese **M-$**. De esta forma se verán algunas opciones en la parte superior de la pantalla, igual que en el modo principal de Ispell!

5.5. Formatos y Gráficos

Si bien los documentos SGML pueden ser limpiamente convertidos a otros muchos formatos, hay un par de consideraciones que hacer si vamos a convertir documentos SGML que involucren algún otro tipo de dato que no sea texto. Para explicar esto, supongamos que estamos manteniendo un documento SGML el cual será distribuido en dos formatos básicos: PostScript y HTML. Si el documento no tiene gráficos no habrá problemas serios al realizar las conversiones. Sin embargo, si los tiene hay que tener en cuenta dos características de los mismos: su tamaño y su formato. En primer lugar tomemos como ejemplo una captura de pantalla de 800x600: la misma podrá mostrarse sin problemas en un archivo HTML, pero no entrará en una hoja A4 en un archivo PS o PDF. El problema con los formatos es un poco más complicado. Por ejemplo, debemos tener en cuenta que en un archivo HTML no podremos mostrar un imagen EPS directamente, así como también, en un archivo PostScript no podremos mostrar una imagen PNG.

Probablemente haya una manera de corregir esta clase de limitaciones en los documentos SGML que involucren algún tipo de programa preprocesador, o algo parecido. Sin embargo no estoy enterado de la existencia de algo parecido. Cuando encuentre la herramienta apropiada, probablemente actualice esta sección del documento. Así y todo, he encontrado un procedimiento simple que simplifica la solución de este problema de formatos:

- Créese un subdirectorio llamado `graphic` en el cual guardaremos todas nuestras imágenes. En dicho subdirectorio, créense otros subdirectorios uno para cada tipo de formato. Por ejemplo, en mi caso, trabajo con Gimp y publico mis trabajos tanto en PostScript como en HTML por lo que tengo tres de estos subdirectorios: `XCF` [14], `EPS` y `PNG`.
- Cuando creemos un gráfico, guardémoslo en los diferentes formatos en cada subdirectorio. Para los formatos XCF y EPS guardemos los archivos con extensión, pero guardemos los archivos PNG sin extensión.
- Si en nuestro documento no especificamos el tipo de archivo, cada rutina de conversión esperará una extensión diferente para las imágenes; excepto por la conversión a HTML la cual tomará el nombre del archivo literalmente. Es por esto que en el punto anterior pedí que se guardaran los archivos PNG sin extensión. De todos modos, el explorador de páginas HTML a usar debería ser capaz de determinar el tipo de archivo. He aquí un ejemplo para insertar una captura de pantalla:

  ```
  <screenshot>
    <screeninfo>Emacs 21 Graphical Menus in X</screeninfo>
      <graphic fileref="graphic/emacs-21-menu-grafico"></graphic>
  </screenshot>
  ```

- Antes de ejecutar la herramienta para la conversión de SGML a otro formato, nos debemos asegurar de tener los archivos de tipo correcto en el directorio `graphic`. Por ejemplo, la conversión a formato PostScript esperará que en dicho directorio halla archivos EPS. Así, no tendremos que modificar la declaración de cada etiqueta `graphic` para cada conversión.

- Dado que nuestros archivos PNG no tienen extensión, el parámetro IMG en el archivo HTML tendrá la misma ruta y nombre de archivo que su ancestro SGML.

5.6. Referencias Cruzadas, Enlaces y URLs

Cada elemento SGML puede tener un ID[15] definido por el usuario. Este ID nos permite hacer referencia al elemento con diferentes propósitos y obtener una salida más coqueta en algunos tipos de archivo que soporten ciertas características. Un ejemplo de esto último puede ser el siguiente. Si estamos publicando nuestros documentos en archivos HTML u otro tipo de archivos que soporten hiperenlaces, puede ser últil publicar enlaces dentro del documento que apunten a otras partes del mismo. Estos enlaces pueden ser declarados de diferentes maneras en los documentos Docbook y en esta sección se darán algunas características que he encontrado dignas de relatar.

5.6.1. XRef

He encontrado a `XRef` muy versátil dado que trabaja de igual forma en publicaciones para la impresión (PostScript) como en publicaciones HTML. La única diferencia es que en este último formato agrega el hipervínculo.

Para usar `XRef` se debe, primero que nada, identificar el nodo objetivo del vínculo (o sea, aquella cosa a la cual vamos a apuntar). Para esto, simplemente deberemos ir a la etiqueta de apertura del elemento que queremos identificar, teclear **C-c +** y seleccionar `ID`. Luego deberemos introducir el nombre:

```
<sect1 id="lfpt">
   <title>La Fantasía de los Procesadores de Texto</title>
```

A partir de ahora, en cualquier lugar podremos hacer referencia a la sección del ejemplo anterior, utilizando su ID de la forma:

```
los cuales fueron listados en <xref linkend="lfpt">
```

5.6.2. URL

Para declarar una URL el procedimiento es bastante trivial. Simplemente debe usarse el elemento `ULink` de modo análogo a como se hace en el siguiente ejemplo:

```
<para>
   <ulink url="http://xml.coverpages.org/general.html#faq">
      http://xml.coverpages.org/general.html#faq
  </ulink>
</para>
```

5.7. Otros trucos bonitos

Aquí cubriré otras características bonitas que he descubierto gracias al uso de Emacs y el módulo PSGML. Esta sección crecerá con el tiempo, por lo que el lector debería mantenerse en línea....

5.7.1. Etiquetado no planeado

No he podido encontrar un mejor título para esta sección por lo que los lectores tendrán que soportar éste. Supóngase que se está realizando la corrección de un documento ya terminado y se decide enfatizar palabras que no lo están. Si se usa **C-c C-e** se terminará con una etiqueta de apertura seguida de una etiqueta de fin pero con nada en el medio. Sin lugar a dudas, esto no es lo que queremos. En este caso se podrá usar **C-<** lo cual nos dejará elegir una etiqueta, pero sólo insertará la de apertura. Cuando se quiera cerrar dicho elemento, se podrá presionar **C-/** ¡lo cual insertará automáticamente en ese lugar la etiqueta de fin!

5.7.2. Parámetros de las Etiquetas

Cuando escribimos en Docbook o en HTML, podemos ver que las etiquetas aceptan gran variedad de parámetros diferentes, los cuales pueden ser muy difíciles de recordar de memoria, o hasta pueden tener valores por defecto. En estos casos, lo que puede hacerse es, ubicando el cursor en cualquier parte de una etiqueta de abertura, presionar **C-c +**. Esto hará que se abra una lista de parámetros válidos en dicha etiqueta y hasta se podrán ver los valores permitidos para cada uno de estos parámetros, si es que han sido definidos. Encuentro esta característica sorprendente y esencial para cualquier edición que involucre etiquetas. Lo que quiero decir, es que ya no necesitaremos tener al lado alguna referencia al lenguaje HTML o Docbook al editar documentos en tales formatos, ya que la mayoría de la información de los parámetros de una etiqueta la podremos obtener con la órden anterior y el autocompletado. De todos modos, esto no es excusa para no aprender el lenguaje estándar de etiquetas con el cual se trabaje, pero sí sirve para no tener que acordarnos de memoria de estas características de menor importancia.

6. Sobre este artículo

Nota: Las direcciones de correos electrónicos que siguen están levemente modificadas para evitar SPAM.

6.1. Sobre el autor

El nombre del autor de este documento es Alejandro Imass, (ait (http://bios.linuxguru.net/view/ait)). Se le puede encontrar en esta dirección (mailto:ait@ tariffi.net) de correo electrónico y en su página web

(http://www.tariffi.net/~ait) se podrán encontrar más detalles sobre él. Su ubicación actual es Guatire, Miranda, Venezuela y es inventor. Escribió este artículo el 12 de Noviembre de 2002.

6.2. Sobre la traducción al español

La traducción al español de este documento fue llevada a cabo por Sebastián Gurin (Cancerbero) a quien se lo puede encontrar mandándole un correo electrónico a su dirección (mailto:cancerbero_sgx@ users.sourceforge.net).

Como todo trabajo humano, la traducción debe tener errores, por lo que el traductor ruega recibir comentarios, sugerencias, retos, etc, de los lectores hispanoparlantes que lean este documento.

Notas

1. WYSIWYG son las siglas de “What You See Is What You Get” (“Lo que ves es lo que obtienes”). Esto hace referencia a la filosofía de editores de textos como MS-Word, Word Perfect, Kword, o Writer de OpenOffice, en los que el usuario debe preocuparse por la apariencia y estructuración del formato del documento.
2. De aquí en adelante se traducirá “major mode” y “minor mode” al español como “modo principal” y “modo secundario”, respectivamente. Nota del Traductor.
3. del término “schemes”. Nota del Traductor.
4. traducción de “tags” al español.
5. http://articles.linuxguru.net/view/202#FTN.AEN17 (http://articles.linuxguru.net/view/202#FTN.AEN17)
6. No olvidemos otro problema bastante grave de los formatos binarios como `.doc`, especialmente grave en nuestros días de correo electrónico. Un archivo binario puede traer embebidas en su código algunas sentencias que pueden ser ejecutadas sin que el usuario se dé cuenta. Esto puede ser utilizado con malas intenciones (especialmente en los sistemas operativos Windows): hay testimonios de contaminación de virus informáticos a través de archivos binarios como los de Office. Actualmente, los antivirus traen opciones para verificar esta clase de documentos antes de abrirlos. Nota del Traductor.
7. creo que dicha razón toma en este caso el nombre de publicidad. Nota del Traductor-Comentarista ;-)
8. Se han traducido “*Text Processing*” y “*Word Processing*” al español como “Formateo de Textos” y “Procesamiento de Textos” respectivamente. Aunque no parezcan las traducciones correctas, existe una diferencia sutil entre ambos conceptos en inglés: “Word Processing” hace referencia a editores WYSIWYG como MS Word mientras que “Text Processing” hace referencia a la edición de documentos en formatos estructurados, p.ej: SGML-Docbook. Sin embargo, empresas como Microsoft han traducido “Word Processing” como “Procesadores de Textos”. Así, si usáramos las traducciones literales de dichos términos causaríamos no poca confusión en los lectores.
9. http://xml.coverpages.org/general.html#faq (http://xml.coverpages.org/general.html#faq) es un buen enlace en donde se detalla la historia de SGML.

10. http://articles.linuxguru.net/view/202#FTN.AEN74 (http://articles.linuxguru.net/view/202#FTN.AEN74)
11. La sutil precisión de la oración tiene como propósito impedir en este documento la confrontación Linux vs. GNU/Linux.
12. El lector debe notar que este artículo no es una guía paso-a-paso sobre el uso de Emacs. Hay mucha información escrita sobre esto. Este artículo se enfocará en el uso del modo principal PSGML y el DTD Docbook. De todos modos se hará una reseña paso-a-paso sobre el conocimiento mínimo necesario para que un principiante pueda manejarse en este documento. El resto debería ser investigado por el lector.
13. que es la traducción al español de “Syntax Highlighting”. Nota del traductor.
14. XCF - Formato Nativo de Gimp.
15. que vendría a ser como un acrónimo de IDentificador.