LinuxDoc+Emacs+Ispell-HOWTO

Auteur : Philippe MARTIN (*feloy@wanadoo.fr*) v0.5, 28 Avril 1998

Ce document s'adresse aux traducteurs et rédacteurs de HOWTO Linux ou de toute autre documentation du Projet de Documentation Linux. Il donne à ces derniers des trucs et astuces pour l'utilisation entre autre d'Emacs et d'Ispell.

Contents

1 Préambule

1.1 Copyright

1.2 Remerciements

Je remercie tout particulièrement Sébastien Blondeel pour m’avoir posé tant de questions si pertinentes sur la configuration d’Emacs, questions qui m’ont permises de concrétiser mes connaissances sur le sujet et de vous en faire part.

1.3 Remarques

N’hésitez pas à me faire parvenir toute remarque et/ou critique sur ce document, je les examinerai afin de faire évoluer au mieux ce document.
N’hésitez pas non plus à me poser des questions relatives au sujet traité dans cet HOWTO, j’y répondrai avec un intérêt tout particulier.

1.4 Versions

Ce document traite des versions suivantes :

- Sgml-tools version 0.99,
- Emacs version 19.34,
- Ispell version 3.1,
- Toutes les bibliothèques Emacs auxquelles je fais référence sont celles distribuées avec la version d’Emacs précitée, sauf `iso-sgml` distribuée avec XEmacs et `psgml` distribuée seule.

2 Introduction

2.1 Le langage SGML

Le langage **SGML** (*Standard Generalized Markup Language*) est un langage qui permet de définir des types de documents.

On peut par exemple définir grâce à lui un type de document *recette de cuisine*, qui précisera qu'il faudra y inclure une première partie avec les différents ingrédients, puis une deuxième avec les accessoires, une troisième donnant la façon de faire votre gâteau pas à pas, et enfin une belle photo qui montre le résultat final.

Cette définition d'un type de document est appelé **DTD** (*Document Type Definition*). Elle ne permet pas de définir quelle sera l'allure du document final, mais seulement ce qu'il contiendra ou pourra contenir.

Pour reprendre l'exemple précédent, je suis sûr qu'en lisant l'idée que je me fais d'une recette de cuisine, vous avez reconnu celles que vous ou votre cuistot attitré utilisez. Et pourtant, elles ont une allure complètement différente : les miennes ont une photo qui se trouve en haut à gauche du placard de la salle de bain, et la liste des ingrédients se trouve entre les bouteilles de lait et de gaz au fond du jardin. Et les vôtres ?

Grâce à cette définition type, quelqu'un peut écrire son document, sans tenir compte de la forme qu'il aura quand il arrivera devant les yeux du lecteur.

2.2 La définition du type `LinuxDoc`

Ce type permet d'écrire, vous l'aviez deviné, des documents se reportant à Linux.

Ces documents sont en général construits ainsi : ils commencent par un titre suivi de l'auteur, de la date de diffusion et de la version de ce document. Suit juste après un court paragraphe qui explique brièvement ce que contient ce document (ce qui évite d'attendre de l'avoir fini pour se rendre compte que ce n'est pas du tout ce que l'on recherchait) puis une table des matières qui montre plus en détail son contenu et permet aux plus pressés d'aller voir directement ce qu'ils cherchent.

Et vient ensuite une suite de chapitres, sections, paragraphes. Au milieu de ces paragraphes, on peut insérer des morceaux de programme, ou changer de police de caractères pour faire ressortir un mot ou un passage, ou encore y insérer des listes, faire référence à une autre partie du document, et d'autres choses encore.

Pour écrire un tel document, il suffit alors de préciser au bon moment le titre, l'auteur, la date et la version du document, puis les chapitres et sections, dire quand il faut insérer une liste et quels en sont les éléments, et ainsi de suite.

2.3 Les `SGML-Tools`

Les **SGML-Tools** permettent, à partir de cette spécification du document, d'obtenir le document final dans le format que vous préférez. Si vous voulez l'ajouter à votre bibliothèque personnelle, ce sera sûrement du *PostScript*, si vous voulez le faire partager au reste de la planète à travers le Web, pourquoi pas du *HTML*, ou si vous craquez et voulez le lire sous Windows, vous pouvez toujours l'avoir en *RTF* pour pouvoir le lire sous n'importe quel éditeur de texte. Ou peut-être sous chacun de ces formats, si vous avez des humeurs changeantes.

La distribution SGML-Tools peut être récupérée via FTP anonyme dans le répertoire *ftp://ftp.lip6.fr/pub/sgml-tools/*

3 Votre premier document

3.1 Depuis un document au format texte

Si vous possédez un document au format texte que vous désirez formater en SGML pour ensuite le transformer en divers formats, voici la marche à suivre :

1. Ajoutez au tout début du fichier les lignes suivantes :

   ```
   <!doctype linuxdoc system>
   <article>
   <title>le titre ici</title>
   <author>
   le nom de l'auteur, son email
   </author>
   <date>
   la version et la date du document
   </date>
   ```

2. S'il y a au tout début du document un court paragraphe qui décrit brièvement le contenu de ce document, encadrez ce paragraphe avec les balises `<abstract>` et `</abstract>`.

3. Ajoutez juste à la suite la balise `<toc>`, qui insère automatiquement la table des matières.

4. Au début de chaque nouveau chapitre, remplacez la ligne contenant le numéro et le titre du chapitre par :

   ```
   <sect>le titre du chapitre
   ```

   et rajoutez la balise `</sect>` à la fin du chapitre.

   Note : Il n'est pas nécessaire d'indiquer le numéro du chapitre, car ils sont numérotés automatiquement.

5. Faites la même chose pour les sections, en remplaçant le numéro et le titre par `<sect1>` et en rajoutant `</sect1>` à la fin de la section.

6. Il est possible de créer des sous-sections jusqu'au niveau 4 (avec `<sect4>` et `</sect4>`) en opérant de la même manière.

7. À chaque début de paragraphe, ajoutez la balise `<p>`.

8. Si vous désirez mettre en valeur certains mots ou certaines phrases, entourez-les des balises `<it>` et `</it>` (*italique*), ou `<bf>` et `</bf>` (**gras**), ou encore `<tt>` et `</tt>` (`caractères de machine à écrire d'antan`).

9. Lorsqu'une liste apparaît dans le texte, comme celle-ci par exemple :

   ```
   Voici une liste :

   - d'une ligne,
   ```

```
- ah non, deux,
- tiens, trois,
- c'est tout.

de quatre lignes !
```

il faut la remplacer par :

```
 Voici une liste :
<itemize>
<item>d'une ligne,
<item>ah non, deux,
<item>tiens, trois,
<item>c'est tout.
 </itemize>
 de quatre lignes !
```

10. Lorsqu'un bloc représente une partie de programme, ou autre chose que l'on veut faire ressortir :

```
<verb>
 10 REM Tiens qu'est-ce que c'est ?
 20 REM Je croyais que ca n'existait plus !
 30 PRINT "Je suis de retour ";
 40 PRINT "pour sauver le monde."
 50 INPUT "Des mains de qui d'apres toi ?",M$
 60 IF M$="Bill" THEN PRINT "Tu es un sage":GOTO AUPARADIS
 70 ELSE PRINT "Tu n'as rien compris...":GOTO AUPAYSDUDOLLAR
 </verb>
```

11. Arrivé à ce point, vous avez déjà bien avancé votre formatage en SGML. Vous pourrez, si vous voulez affiner votre document, jeter un oeil sur le guide d'utilisation des **SGML-Tools**, qui décrit plus en détail le type de document **LinuxDoc**.

4 Configurer Emacs

4.1 Les caractères accentués

Si vous désirez écrire des documents en français ou dans une autre langue européenne, vous aurez besoin de caractères accentués. Voici comment configurer Emacs pour qu'il accepte ces caractères.

4.1.1 L'affichage des caractères 8 bits

Pour qu'Emacs soit capable d'afficher des caractères 8 bits, ajoutez les lignes suivantes à votre `.emacs` :

```
(standard-display-european 1)
(load-library "iso-syntax")
```

Si vous utilisez Emacs sur un terminal qui ne supporte pas l'affichage des caractères 8 bits, vous pouvez utiliser la bibliothèque `iso-ascii` (`(load-library "iso-ascii")`), qui permet à Emacs d'afficher les caractères 8 bits d'une façon approchée.

4.1.2 La saisie des caractères accentués

Si votre clavier permet de taper les caractères accentués, aucun problème ne devrait se poser. En revanche, s'il ne le permet pas, voici quelques moyens d'y remédier :

La bibliothèque `iso-acc` La bibliothèque `iso-acc` d'Emacs permet d'entrer des caractères accentués malgré que l'on ait un clavier qui ne le permette a priori pas.

Pour utiliser cette bibliothèque, ajoutez la ligne suivante à votre `.emacs` :

```
(load-library "iso-acc")
```

Puis, une fois avoir relancé Emacs et ouvert le fichier que vous voulez éditer, tapez `Meta-x iso-accents-mode`.

Vous pouvez alors entrer un caractère `é` en tapant `'` puis `e`. De manière générale, on peut entrer un caractère accentué en tapant d'abord l'accent, puis la lettre à accentuer (aussi bien minuscule que majuscule). Voici les différents accents utilisables :

- `'` : Un accent aigu
- `` ` `` : Un accent grave
- `^` : Un accent circonflexe
- `"` : Un tréma
- `~` : Un tilde au dessus de la lettre, une cédille lorsqu'il précède un c, et d'autres encore (voir fichier iso-acc.el),
- `/` : Pour barrer une lettre, ...

Si vous avez besoin d'entrer un de ces caractères et non pas une lettre accentuée, tapez un espace à la suite de l'accent. Par exemple, pour taper *l'éléphant*, tapez `l '` ⟨*spc*⟩ `' e l ' e ...`

Vous pouvez trouver l'ensemble des combinaisons dans le fichier `iso-acc.el`.

La touche ⟨*Meta*⟩ Il est possible avec certains terminaux de saisir des caractères accentués grâce à la touche ⟨*Meta*⟩ (ou ⟨*Alt*⟩). Par exemple, la frappe de ⟨*Meta*⟩**-i** permet d'entrer le caractère **é**.

Mais Emacs prévoit la touche ⟨*Meta*⟩ pour d'autres utilisations. Il faut donc taper la séquence **Ctrl-q** (commande `quoted-insert`) avant de taper ⟨*Meta*⟩**-i**.

Cette commande permet aussi d'entrer un caractère selon son code en octal. Tapez **Ctrl-q** suivi du code en octal du caractère que vous désirez entrer.

4.1.3 Afficher les caractères SGML en 8 bits

En SGML, les caractères accentués peuvent être tapés grâce à des macros. Par exemple, le caractère **é** s'écrit **é**. En général, les applications qui lisent du SGML arrivent à lire les caractères 8 bits et il n'est donc pas nécessaire d'utiliser ces macros. Mais il est possible que certaines ne le puissent pas. En sachant qu'il existe un moyen facile de remédier à ça, ce serait dommage de faire "planter" ces dernières applications.

En effet, la bibliothèque `iso-sgml` vous permet d'entrer sous Emacs des caractères accentués, comme d'habitude, mais lorsqu'il enregistre votre fichier, il transforme tous ces caractères 8 bits par leur équivalent SGML.

Il est donc très facile, grâce à cette bibliothèque, de saisir et relire votre document sous Emacs, et vous êtes sûr que votre document ne sera pas rejeté par une application qui ne comprend pas les caractères 8 bits.

Pour utiliser cette bibliothèque, il suffit de rajouter ces lignes à votre `.emacs` :

```
(setq sgml-mode-hook
'(lambda () "Defauts pour le mode SGML."
(load-library "iso-sgml")))
```

4.2 Le mode SGML

Lorsque vous ouvrez un fichier avec une extension **.sgml**, Emacs lance normalement le **mode sgml**. S'il ne le fait pas, vous pouvez soit le faire manuellement avec `Meta-x sgml-mode`, soit le lancer automatiquement en rajoutant ces lignes à votre `.emacs` :

```
(setq auto-mode-alist
(append '(("\.sgml$"  . sgml-mode))
          auto-mode-alist))
```

Ce mode permet par exemple de choisir comment insérer les caractères 8 bits. En utilisant `Meta-x sgml-name-8bit-mode` (ou dans le menu *SGML/Toggle 8 bit insertion*), vous pouvez choisir d'insérer les caractères 8 bits soit tels quels, soit sous leur forme SGML, c'est-à-dire sous la forme **&...;**.

Il permet aussi de cacher ou non les balises SGML, avec `Meta-x sgml-tags-invisible` ou le menu *SGML/Toggle Tag Visibility*.

4.3 Le mode PSGML

Le mode PSGML donne de grandes facilités pour éditer des documents SGML sous Emacs.

La documentation *psgml-linuxdoc*

explique comment installer et utiliser ce mode conjointement avec *LinuxDoc*.

4.4 Divers

4.4.1 Le mode auto-fill

En mode normal, lorsque vous tapez un paragraphe et que vous arrivez en bout de ligne, vous devez vous-même utiliser la touche ⟨*Entrée*⟩ pour revenir à la ligne, ou bien votre ligne continue indéfiniment tout le long du paragraphe. Il en résulte, si vous utilisez ⟨*Entrée*⟩ pour revenir à la ligne, un paragraphe dont les fins de lignes ne sont pas alignées, et en général ces lignes rallongent ou raccourcissent au fur et à mesure. De même si vous laissez des lignes dépasser une longueur raisonnable, vous ne pourrez pas les voir sous certains éditeurs.

Le mode **auto-fill** permet d'automatiser cette tâche ingrate : lorsque vous dépassez une certaine colonne (la 70e par défaut), il vous place automatiquement à la ligne suivante.

Voici comment utiliser ce mode, et fixer la largeur de vos lignes à 80 :

```
(setq sgml-mode-hook
'(lambda () "Defauts pour le mode SGML."
(auto-fill-mode)
(setq fill-column 80)))
```

5 Ispell

Si vous désirez vérifier l'orthographe de votre document directement depuis Emacs, vous pouvez utiliser la distribution **Ispell** et son mode associé sous Emacs.

5.1 Configuration initiale

Tout d'abord, ajoutez ces lignes à votre `.emacs` pour configurer Emacs :

```
(autoload 'ispell-word "ispell"
   "Check the spelling of word in buffer." t)
(global-set-key "\e$" 'ispell-word)
(autoload 'ispell-region "ispell"
   "Check the spelling of region." t)
(autoload 'ispell-buffer "ispell"
   "Check the spelling of buffer." t)
(autoload 'ispell-complete-word "ispell"
   "Look up current word in dictionary and try to complete it." t)
(autoload 'ispell-change-dictionary "ispell"
   "Change ispell dictionary." t)
(autoload 'ispell-message "ispell"
   "Check spelling of mail message or news post.")
(autoload 'ispell-minor-mode "ispell"
   "Toggle mode to automatically spell check words as they are typed in.")
```

5.2 Le choix de vos dictionnaires par défaut

Vous pouvez configurer Emacs pour qu'à l'ouverture d'un fichier, celui-ci choisisse automatiquement quels dictionnaires utiliser. Vous pouvez en effet utiliser plusieurs dictionnaires. Le premier et sûrement le plus important est le dictionnaire principal, distribué avec Ispell. Vous avez le choix entre plusieurs langues. Le deuxième est votre dictionnaire personnel, celui où Ispell ajoutera les mots qu'il n'aura pas trouvé dans le premier dictionnaire mais que vous lui aurez indiqué de garder.

Voici les lignes à insérer à votre `.emacs` si vous désirez utiliser par défaut le dictionnaire français distribué avec Ispell, et placer votre dictionnaire personnel dans un fichier `.ispell-dico-perso` dans votre répertoire racine.

```
 (setq sgml-mode-hook
'(lambda () "Defauts pour le mode SGML."
(setq ispell-personal-dictionary "~/.ispell-dico-perso")
(ispell-change-dictionary "francais")
 ))
```

5.3 Choix des dictionnaires pour un certain fichier

Un petit problème se pose si vous ne vérifiez pas toujours des textes dans la même langue. Et si vous traduisez des documents, il est probable que vous passiez d'une langue à l'autre assez souvent.

Je ne connais pas de moyen en Lisp de choisir, soit automatiquement, soit en un *click* de souris, les dictionnaires principaux et personnels associés à la langue utilisée dans le fichier en cours. (Si vous en connaissez un, faites-moi signe !)

Mais il est possible d'indiquer quels dictionnaires vous voulez utiliser pour ce fichier (et seulement celui-là). Il suffit de les rajouter en commentaire, à la fin du fichier, pour qu'Ispell puisse les lire en lançant une vérification :

```
<!-- Local IspellDict: english -->
<!-- Local IspellPersDict: ~/emacs/.ispell-english -->
```

Si vous avez défini dans votre `.emacs` que vos dictionnaires par défaut seront français, vous pouvez alors ajouter ces lignes à chaque fin de fichier dont le texte est en anglais.

5.4 Vérifier votre document

Pour lancer la vérification de votre document en intégralité, utilisez, depuis n'importe où dans votre document `Meta-x ispell-buffer`. Vous pouvez aussi lancer la vérification sur une région seulement du document :

- Indiquez le début de la région avec `Ctrl-Spc` (mark-set-command),
- Placez-vous à la fin de la région à vérifier,
- tapez `Meta-x ispell-region`.

Emacs lance alors Ispell. Si ce dernier trouve un mot qu'il ne connait pas, il vous indique ce mot (normalement en surbrillance) et vous demande de presser une touche :

- **spc** accepte ce mot, uniquement pour cette fois,
- **i** accepte ce mot et l'insère dans votre dictionnaire personnel,
- **a** accepte ce mot pour cette session,
- **A** accepte ce mot pour ce fichier, en l'insérant dans le dictionnaire local au fichier,

- **r** permet de corriger le mot mal orthographié,
- **R** permet de corriger toutes les occurrences du mot mal orthographié,
- **x** arrête la vérification, et replace le curseur à sa position initiale,
- **X** arrête la vérification en laissant le curseur sur le mot mal orthographié, vous permettant de modifier votre fichier ; vous pouvez continuer la vérification en tapant `Meta-x ispell-continue`,
- **?** affiche une aide en ligne.

Si Ispell trouve un ou plusieurs mots ressemblant à celui qu'il ne connait pas, il vous les indique dans une petite fenêtre, chacun précédé d'un chiffre. Il suffit de presser un de ces chiffres pour corriger le mot mal orthographié par le mot correspondant.

5.5 Dictionnaire personnel contre dictionnaire local au fichier

La touche **i** permet d'insérer un mot dans le dictionnaire personnel, alors que la touche **A** permet d'insérer un mot dans le dictionnaire local au fichier.

Le dictionnaire local au fichier est une suite de mots insérés à la fin du fichier, sous forme de commentaires, et qui est relu par Ispell chaque fois que vous le lancez sur ce fichier. Cela permet d'accepter certains mots, valables dans ce fichier, mais qui ne le seraient pas dans d'autres.

A mon avis, il est préférable que le dictionnaire personnel soit réservé aux mots que le dictionnaire principal ne connait pas mais qui font vraiment partie de la langue (comme les mots composés), plus certains mots n'appartenant pas à la langue ou noms propres qui reviennent dans un grand nombre de fichiers (comme *Linux*) et ne ressemblant pas trop à un mot du dictionnaire principal : l'ajout par exemple de certains noms et prénoms de personnes dans le dictionnaire personnel peut être dangereux, car ils ressemblent parfois à un mot de la langue (imaginez qu'il ne trouve pas de fautes dans la phrase : '*Ted en est l'effet au phil du temps.*' !).

5.6 La vérification 'à la volée'

Ispell peut aussi vérifier l'orthographe au fur et à mesure que vous tapez votre document. Il faut utiliser pour cela le mode **ispell-minor-mode**. Lorsque vous désirez lancer ou arrêter ce mode de vérification, tapez `Meta-x ispell-minor-mode`. Ispell vous envoie alors un *bip* chaque fois que vous tapez un mot qu'il ne connait pas.

Si ces *bip* à répétition vous ennuient (ou si votre voisin de palier dort !), vous pouvez les remplacer par un flash de l'écran, en tapant `Meta-x set-variable RET visible-bell RET t RET`. Ou ajoutez cette ligne à votre `.emacs` pour faire taire Emacs à tout jamais :

```
(setq visible-bell t)
```

5.7 Saut de régions

Il est possible de ne pas vérifier votre document en intégralité, en sautant quelques régions bien spécifiques. En effet, vous voudrez sûrement ne pas vérifier l'orthographe de vos balises SGML. Pour cela, ajoutez la ligne suivante à votre `.emacs` :

```
(setq ispell-skip-sgml t)
```

Une version Beta du 20 Mars 98 du fichier `ispell.el` est disponible à l'adresse *http://kdstevens.com/~stevens/ispell-page.html* . Cette version étend les régions SGML à sauter. En effet, la version d'`ispell.el` fournie avec Emacs permet seulement de sauter les tags SGML, de la forme <**tt**> ou </**tt**>.

Cette version Beta permet de sauter également les régions entre :

- <author> et la fin de la ligne,
- < et / (pour les tags SGML de la forme <**em/...**/,
- <code> et </code>,
- <verb> et </verb>,
- <tt> et </tt>.

6 Pour aller plus loin

6.1 Insertion automatique d'une entête

Sous Emacs, il est possible d'*accrocher* des actions à chaque événement (ouverture d'un fichier, sauvegarde, lancement d'un mode, etc).

La bibliothèque **autoinsert** utilise cette fonctionnalité : lorsque vous ouvrez un nouveau fichier sous Emacs, cette bibliothèque insère, selon le type de ce fichier, une entête *standard*.

Dans notre cas, cette entête *standard* pourrait bien être la partie qui déclare le type de document (LinuxDoc), le titre, l'auteur et la date.

Je vais décrire ici deux façons d'insérer une telle entête. Soit en insérant un fichier que vous aurez préalablement écrit, soit en lançant une routine écrite en **elisp**.

6.1.1 par l'insertion d'un fichier

Il faut tout d'abord préciser à Emacs d'exécuter la commande `auto-insert` à l'ouverture d'un fichier, puis lire la bibliothèque **autoinsert** qui déclare la liste `auto-insert-alist` qu'il nous faut modifier, cette dernière définissant pour chaque type de fichier l'entête à insérer. Le fichier à insérer doit par défaut se trouver dans le répertoire `~/insert/`, mais il est possible de redéfinir la variable `auto-insert-directory` si l'on veut le placer ailleurs.

Voici les lignes à rajouter à votre `.emacs` pour insérer le fichier `~/emacs/sgml-insert.sgml` à l'ouverture d'un nouveau fichier SGML :

```
(add-hook 'find-file-hooks 'auto-insert)
(load-library "autoinsert")
(setq auto-insert-directory "~/emacs/")
(setq auto-insert-alist
      (append '((sgml-mode .  "sgml-insert.sgml"))
              auto-insert-alist))
```

Vous pouvez alors écrire dans le fichier `~/emacs/sgml-insert.sgml` votre entête personnalisée, puis relancer Emacs et ouvrir un fichier `toto.sgml`. Emacs devrait alors vous demander de confirmer l'insertion automatique, et dans l'affirmative insérer votre entête.

6.1.2 par l'exécution d'une routine

Cela fonctionne un peu comme précédemment, mais au lieu de préciser dans la variable `auto-insert-alist` un fichier à insérer, il faut préciser une fonction à exécuter. Voici comment procéder, en supposant que l'on écrive cette fonction dans un fichier `~/emacs/sgml-header.el` (inutile d'encombrer votre `.emacs` avec cette fonction qui peut se révéler assez longue) :

```
(add-hook 'find-file-hooks 'auto-insert)
(load-library "autoinsert")
(add-to-list 'load-path "~/emacs")
(load-library "sgml-header")
(setq auto-insert-alist
      (append '(((sgml-mode .  "SGML Mode") . insert-sgml-header))
              auto-insert-alist))
```

Vous pourrez trouver en A (appendice) un exemple de cette fonction `insert-sgml-header`.

A Une fonction insert-sgml-header

Cette fonction permet à l'utilisateur d'insérer une entête personnalisée pour un document du Projet de Documentation Linux dans un fichier. Elle peut soit être appelée automatiquement lorsque l'on ouvre un nouveau fichier SGML, soit être appelée explicitement par l'utilisateur.

Cette fonction demande à l'utilisateur, à travers le *mini-buffer*, divers renseignements, certains nécessaires, d'autres facultatifs.

Tout d'abord le titre. Si l'utilisateur n'entre aucun titre, la fonction retourne immédiatement, et rien n'est inséré. Vient ensuite la date, l'auteur, son email et sa home page (ces deux derniers étant facultatifs).

Il demande ensuite le nom du traducteur. S'il n'y a pas de traducteur pour ce document, il suffit de presser *Entrée*, et les renseignements sur le traducteur ne seront pas demandés. Dans le cas contraire, la fonction demande l'email et la home page du traducteur (aussi facultatifs).

Cette fonction affiche alors dans le buffer courant votre entête, comportant bien sûr tous les renseignements que vous avez cités mis en forme, mais aussi les balises nécessaires au paragraphe d'introduction et au premier chapitre, puis place le curseur à l'endroit où vous pourrez saisir le paragraphe d'introduction.

```
(defun insert-sgml-header ()
  "Insere l'entete d'un document LinuxDoc"
  (interactive)
  (let (titre auteur email home traducteur email-traducteur home-traducteur date
              point-debut)
    (setq titre (read-from-minibuffer "Titre : "))
    (if (> (length titre) 0)
        (progn
          (setq date (read-from-minibuffer "Date : ")
                auteur (read-from-minibuffer "Auteur : ")
```

```
      email (read-from-minibuffer "Email auteur : ")
      home (read-from-minibuffer "Home page auteur : http://")
      traducteur (read-from-minibuffer "Traducteur : "))
(insert "<!doctype linuxdoc system>\n<article>\n<title>")
(insert titre)
(insert "</title>\n<author>\nAuteur : ") (insert auteur) (insert "<newline>\n")
(if (> (length email) 0)
    (progn
      (insert "<htmlurl url=\"mailto:")
      (insert email) (insert "\" name=\"") (insert email)
      (insert "\"><newline>\n")))
(if (> (length home) 0)
    (progn
      (insert "<htmlurl url=\"http://")
      (insert home) (insert "\" name=\"") (insert home)
      (insert "\">\n<newline>")))
(if (> (length traducteur) 0)
    (progn
      (setq email-traducteur (read-from-minibuffer "Email traducteur : ")
            home-traducteur (read-from-minibuffer "Home page traducteur : http://"))
      (insert "Traducteur : ")
      (insert traducteur)
      (insert "<newline>\n")
      (if (> (length email-traducteur) 0)
          (progn
            (insert "<htmlurl url=\"mailto:")
            (insert email-traducteur) (insert "\" name=\"")
            (insert email-traducteur)
            (insert "\"><newline>\n")))
      (if (> (length home-traducteur) 0)
          (progn
            (insert "<htmlurl url=\"http://")
            (insert home-traducteur) (insert "\" name=\"")
            (insert home-traducteur)
            (insert "\"><newline>\n")))))
(insert "</author>\n<date>\n")
(insert date)
(insert "\n</date>\n\n<abstract>\n")
(setq point-debut (point))
(insert "\n</abstract>\n<toc>\n\n<sect>\n<p>\n\n\n</sect>\n\n</article>\n")
(goto-char point-debut)
))))
```