Cahiers GUTenberg

COMMENT COMMENTER ? COMMENTAIRES ET PARTIES OPTIONNELLES

Yvon Henel

Cahiers GUTenberg, nº 44-45 (2004), p. 54-82.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_2004___44-45_54_0>

Comment commenter ? Commentaires et parties optionnelles

Yvon HENEL, dit le TEXnicien de surface

Résumé. Comment placer des commentaires dans son texte source, comment traiter des parties optionnelles et obtenir un document multiforme.

Présentation des modules `verbatim`, `comment`, `versions` et `optional`.

Exemples d'utilisation de `docstrip`.

1 Introduction

Quelques remarques d'ordre terminologique tout d'abord. J'appelle « source » le fichier, portant généralement l'extension `tex`, que l'on tape avec son éditeur de texte préféré et que l'on donne à TEX ou un de ses descendants pour qu'il le compile c-à-d. pour qu'il le transforme en un document lisible à l'écran, imprimable, &c. Je parlerai du résultat de la compilation sous le nom de « document final ».

J'appelle « commentaire » tout texte que l'on veut voir dans le source mais qui ne doit pas être visible dans le document final. Il faut bien entendu que la présence du commentaire n'interfère en rien avec le travail de mise en page de TEX.

Je parle de « texte optionnel » pour désigner des fragments de document saisis dans le source et que l'on pourra faire apparaître ou non à l'aide d'une opération simple, relevant de la syntaxe de TEX ou de LATEX, comme le choix d'une option à l'appel d'un module ou l'utilisation d'une commande.

Je me propose de montrer quelques techniques de base pour que du texte saisi dans le source soit compilé ou pas, c-à-d. apparaisse ou non dans le document final.

Tout débutant apprend assez rapidement qu'il est possible de placer des commentaires dans le source à l'aide du caractère % et les exemples plus ou moins humoristiques abondent dans tous les manuels de TEX ou LATEX. Il existe cependant des techniques plus subtiles que je présenterai dans un premier temps.

Ces techniques donnent la possibilité de changer *une* information en début de document pour qu'un certain nombre de textes apparaissent ou disparaissent du

document final ce qui nous amènera tout doucement à la notion de partie optionnelle.

Je présenterai alors quelques modules présents sur CTAN dont c'est précisément l'objet. Je finirai par l'utilisation de `docstrip` dont ce n'est pas directement le propos mais qui permet quelques manipulations parfois utiles.

2 Sans module particulier

Il s'agit donc d'exposer les moyens d'écrire un bout de texte dans le source qui n'apparaîtra pas dans le document final. En général, on veut garder la trace d'une modification, rappeler une source, placer un mot clé que l'on retrouvera aisément avec son éditeur…

Commençons par passer en revue les moyens qui existent sans faire appel à un module spécialisé.

2.1 Le caractère « pour cent »

Je ne pense pas trahir un secret des dieux en disant que le source suivant [1] :

1 - %
```
1 J'écris ceci par une nuit de tempête. % Il est neuf heures !
```

donnera :

J'écris ceci par une nuit de tempête.

C'est la solution la plus simple lorsqu'il s'agit de placer un commentaire court dans le source. Cela devient plus problématique lorsque l'on veut s'en servir pour des portions plus longues, à moins que l'éditeur ne gère cela de manière satisfaisante. Certains d'entre eux en effet savent que faire quand un saut de ligne intervient au milieu d'un commentaire et réagissent en conséquence.

Cependant il peut arriver que l'éditeur se trompe dans certaines situations délicates, je pense au *verbatim*, et il arrive que plusieurs éditeurs n'aient pas la même conception du beau : on tape son source avec l'un et, pressé, on l'ouvre avec un autre plus léger pour une petite correction quelque part ; mais le second, sans rien en dire, recoupe les lignes à son goût et ne tient pas compte correctement des commentaires.

1. La présentation des exemples – lignes de code numérotées et résultat produit, tous deux encadrés – est faite grâce à l'extension `fancyvrb.sty`.

Résultat : des morceaux de textes censés être cachés apparaissent tout soudain dans le document final !

Il est, par ailleurs, assez pénible de devoir chercher dans un document assez long, tout ce qu'il faut « décommenter » pour obtenir la version longue du document dont on vient d'imprimer la version courte. Il faudrait faire appel à un outil extérieur ou utiliser des macros de l'éditeur en n'oubliant pas que l'on a peut-être commenté un commentaire.

On réservera donc l'usage du % à ce pour quoi il a été conçu et on va tenter de trouver plus puissant et plus facile d'emploi.

2.2 Avec des si...

TEX, le langage, contient ce qu'il faut pour écrire des alternatives sous la forme générale `\ifMACHIN` action si-oui `\else` action si-non `\fi`. Je ne tenterai pas ici de rentrer dans le décryptage du mécanisme général mais je montrerai comment utiliser cette « fonctionnalité » pour obtenir ce que je veux, à savoir un texte optionnel.

2.2.1 \iffalse

Tout d'abord TEX propose la commande `\iffalse` qui permet ceci :

2 - \iffalse

```
Un peu de texte
\iffalse
tu parles !
\fi
et encore du texte.
```

Un peu de texte et encore du texte.

On fera attention aux espaces puisque `\fi`, étant une macro, mange l'espace qui la suit et que l'on pourrait avoir des surprises :

3 - pas bon !

```
Un peu de texte\iffalse tu parles !\fi et encore du texte.
```

Un peu de texteet encore du texte.

On a donc un mécanisme qui permet de « commenter » un morceau important du texte et robuste dans le sens où on voit mal quelle manipulation de l'éditeur pourrait le mettre à mal.

Il est possible assez facilement alors de commenter, à l'aide de %, le couple `\iffalse`, `\fi` pour décommenter le commentaire. (Vous suivez ? Voyez ci-avant l'exemple 2.) En général un éditeur un peu évolué proposera des mécanismes qui pourront faire cela assez aisément mais nous n'avons pas encore obtenu ce que nous voulions : un mécanisme purement TEXnique.

2.2.2 \newif

TEX fournit une commande permettant de créer de « nouveaux si ». Avec `\newif\ifMonCom`, je — enfin TEX — crée plusieurs commandes : `\MonComtrue`, son pendant, et opposé, `\MonComfalse` ainsi que `\ifMonCom` qui permet de tester l'état du « booléen » `MonCom`.

On pourra alors faire ceci :

4 - mon si (vrai)

```
\newif\ifPublic \Publictrue
Le bénéfice de notre entreprise est
\ifPublic en hausse \else inexistant \fi
cette année.
```

dont la version — pour les actionnaires de *LaTeX incorporated* — sera :

Le bénéfice de notre entreprise est en hausse cette année.

et la version interne (ou destinée au fisc) :

5 - mon si (faux)

```
\newif\ifPublic \Publicfalse
Le bénéfice de notre entreprise est
\ifPublic en hausse \else inexistant \fi
cette année.
```

Le bénéfice de notre entreprise est inexistant cette année.

Bien entendu, la commande `\Publictrue` ou `\Publicfalse` sera placée en tête du document — après le `\begin{document}` — et on aura placé le `\newif\ifPublic` soit dans le préambule, soit juste avant la commande qui en fixe la valeur de vérité (`\Publictrue` ou `\Publicfalse` selon le cas).

La deuxième branche de l'alternative est optionnelle est on pourra avoir plus simplement :

6 - absurde !
```
\newif\ifMonCom \MonComfalse
Mes mémoires permettront de rétablir la vérité
\ifMonCom ne pas montrer ça à ma femme !\fi
quant aux allégations\dots
```

Mes mémoires permettront de rétablir la vérité quant aux allégations...

3 Des « si » à la LaTeX, le module `ifthen`

3.1 Présentation du module

Le module `ifthen`, écrit à l'origine par Leslie Lamport et réécrit depuis par David Carlisle, fournit quelques commandes bien utiles.

On dispose de la macro `\ifthenelse` qui permet d'écrire, à la LaTeX, une alternative avec `\ifthenelse{<`*test*`>}{<`*action si vrai*`>}{<`*action si faux*`>}`.

Le test peut porter sur l'égalité entre deux chaînes de caractères avec la commande `\equal` comme ceci `\equal{<`*chaîne 1*`>}{<`*chaîne 2*`>}` qui ne sera vrai que si les deux chaînes sont identiques après expansion complète si nécessaire.

Supposons maintenant qu'on veuille créer une variable booléenne nommée `MonBoul` : il nous suffit de déclarer `\newboolean\MonBoul`. La valeur de `MonBoul` pourra être fixée à `true` avec `\setboolean{MonBoul}{true}`.

Cependant il faudra utiliser `\boolean{MonBoul}` comme premier argument de `\ifthenelse` et non pas tout simplement `MonBoul`. On écrira donc

7
```
\ifthenelse{\boolean{MonBoul}}{action si vrai}{action si faux}
```

Il est également possible de faire du calcul avec les booléens. Le module définit les commandes `\and`, `\or` et `\not` ainsi que `\(` et `\)` pour parenthéser les calculs. *Je renvoie le lecteur à la documentation du module pour les autres commandes.*

En supposant que nous ayons défini les variables booléennes correctement nous pourrions donc écrire des choses comme :

8
```
\(\boolean{moi} \or \boolean{lui}\) \and \not\boolean{resume}
```

ou encore

9
```
\(\boolean{moi} \and \boolean{court}\) \or \(\boolean{boss} \and
\boolean{long}\)
```

en 1er argument de `\ifthenelse`.

On peut imbriquer les commandes et écrire quelque chose comme

10
```
\ifthenelse{\boolean{court}}
{Le texte court%
  \ifthenelse{\boolean{notes}}{ [un commentaire pour moi]}{}.}
{Le texte long%
  \ifthenelse{\boolean{notes}}{ [un autre commentaire]}{}
qui continue ici.}
```

Là encore il faudra prendre garde aux espaces de toutes sortes qui pourraient se glisser subrepticement dans le texte. Il faudra ne pas oublier que TEX mange les espaces de début de ligne, ainsi les deux lignes de source suivantes sont rendues identiquement

11
```
Le début de notre histoire
        Le début de notre histoire
```

Il faudra ne pas oublier non plus que la ligne

12
```
\ifthenelse{\boolean{notes}}{ [un autre commentaire pour moi]}{}
```

donne un espace puisque l'on n'a pas « camouflé » le saut de ligne avec %.

3.2 Quelques commandes

On pourra créer quelques commandes pour réaliser notre idée première. En voici deux exemples. Dans le premier on crée un booléen et les deux commandes `\Notes` et `\PiedNotes` y font appel. L'apparition du texte dans le document final est régie par les commandes antithétiques `\AvecNotes` et `\SansNotes` *dont j'espère que les noms seront aisément retenus.* La présentation de la note dans l'exemple n'est due qu'au fait que j'utilise un environnement `minipage` pour écrire l'exemple.

13 - avec un booléen

```
\newboolean{BNotes}\setboolean{BNotes}{false}
\newcommand{\AvecNotes}{\setboolean{BNotes}{true}}
\newcommand{\SansNotes}{\setboolean{BNotes}{false}}
\newcommand{\Notes}[1]{%
  \ifthenelse{\boolean{BNotes}}{ [\emph{#1}]}{}}
\newcommand{\PiedNotes}[1]{%
  \ifthenelse{\boolean{BNotes}}{\footnote{#1}}{}}

\AvecNotes
Un peu de texte\Notes{Tu parles !} et la suite\PiedNotes{Qui
n'en finit pas}.

\SansNotes
Un peu de texte\Notes{Tu parles !} et la suite\PiedNotes{Qui
n'en finit pas}.
```

Un peu de texte [*Tu parles !*] et la suite [a].
Un peu de texte et la suite.

a. Qui n'en finit pas

On peut préférer le deuxième où j'utilise deux commandes qui s'expansent en une chaîne de caractères. Bien entendu on peut remplacer le « non » de `\SansLesNotes` par n'importe quoi différent de « oui ».

14 - avec une chaîne

```
\newcommand{\LesNotes}{non}
\newcommand{\AvecLesNotes}{\renewcommand{\LesNotes}{oui}}
\newcommand{\SansLesNotes}{\renewcommand{\LesNotes}{non}}
\newcommand{\MesNotes}[1]{%
  \ifthenelse{\equal{\LesNotes}{oui}}{ [\emph{#1}]}{}}

\AvecLesNotes
Un peu de texte\MesNotes{Tu parles !} et la suite\MesNotes{Qui
n'en finit pas}.

\SansLesNotes
Un peu de texte\MesNotes{Tu parles !} et la suite\MesNotes{Qui
n'en finit pas}.
```

Un peu de texte [*Tu parles !*] et la suite [*Qui n'en finit pas*].
Un peu de texte et la suite.

3.3 Petits modules faits à la main

Si l'on devait utiliser tout cela de manière récurrente, il conviendrait d'en faire un petit module. En voici le code minimaliste.

15 - mesnotes.sty

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\def\fileversion{v0}
\def\filedate{2004/06/30}
\def\fileinfo{fichier mesnotes.sty par Le TeXnicien de surface}
\ProvidesPackage{mesnotes}
   [\filedate\space\fileversion\space\fileinfo]
\RequirePackage{ifthen}
\newboolean{BNotes}\setboolean{BNotes}{false}
\newcommand{\AvecNotes}{\setboolean{BNotes}{true}}
\newcommand{\SansNotes}{\setboolean{BNotes}{false}}
\newcommand{\Notes}[1]{%
  \ifthenelse{\boolean{BNotes}}{ [\emph{#1}]}{}}
\newcommand{\PiedNotes}[1]{%
  \ifthenelse{\boolean{BNotes}}{\footnote{#1}}{}}
\endinput
```

On pourra alors l'appeler avec `\usepackage{mesnotes}` et placer `\AvecNotes` ou `\SansNotes` dans le préambule ou en tête du source.

Une autre solution s'offre encore à nous : créer un module avec des options qui régleraient l'apparition ou non des notes. C'est l'objet de ce qui suit.

16 avec options

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\def\fileversion{v1}
\def\filedate{2004/06/30}
\def\fileinfo{fichier mesnotes.sty par Le TeXnicien de surface}
\ProvidesPackage{mesnotes}
  [\filedate\space\fileversion\space\fileinfo]
  \RequirePackage{ifthen}
  \newboolean{BNotes}
  \DeclareOption{avec}{\setboolean{BNotes}{true}}
  \DeclareOption{sans}{\setboolean{BNotes}{false}}
  \DeclareOption*{\PackageError{mesnotes}{%
      L'option \CurrentOption\space n'est pas connue !
      \MessageBreak
      C'est "sans" qui est l'option par défaut.
    }{Choisissez entre "sans"
```

```
16      (pour que les notes restent muettes)
17      \MessageBreak et "avec"
18      (pour que les notes apparaissent)}}
19  \ExecuteOptions{sans}
20  \ProcessOptions\relax
21  \newcommand{\Notes}[1]{%
22    \ifthenelse{\boolean{BNotes}}{ [\emph{#1}]}{}}
23  \newcommand{\PiedNotes}[1]{%
24    \ifthenelse{\boolean{BNotes}}{\footnote{#1}}{}}
25  \endinput
```

La ligne 11 permet d'intercepter les options non reconnues et de définir le message d'erreur puis de donner des précisions lorsque l'utilisateur tape « h » à la question de TEX. La ligne 19 définit l'option par défaut : sans option explicite, le module empêchera l'apparition des notes dans le document final.

On aura donc le même comportement avec `\usepackage{mesnotes}` ou `\usepackage[sans]{mesnotes}`. Pour faire apparaître les notes, placées dans le source avec `\Notes` on changera l'option `sans` en `avec` et la nouvelle compilation nous donnera ce que l'on attend.

Il reste cependant un petit problème avec la commande `\Notes` telle qu'elle est écrite aux lignes 21 et 22. En effet dans l'état elle nous force à faire attention aux espaces qui précèdent et qui suivent car sa présence empêche TEX de fusionner les espaces qu'elle sépare et l'on peut se retrouver avec une double espace inter-mot dans le document final lorsque les commentaires sont cachés. Il existe cependant deux commandes définies dans les sources de LATEX qui permettent de pallier cet inconvénient. Les voici dans une définition plus astucieuse et robuste de la commande `\Notes`.

17 - notes améliorées

```
1  \newcommand{\Notes}[1]{%
2  \@bsphack \ifthenelse{\boolean{BNotes}}{ [\emph{#1}] }{}%
3  \@esphack}
```

On prendra garde à l'espace dans `{[\emph{#1}]␣}` pour que le texte qui suit ne se colle pas à l'accolade fermante. On a ôté celui qui précédait l'accolade ouvrante pour éviter la double espace dans le document final lorsque la note apparaît.

4 Prêt à manger

Ce nom d'une chaîne de « restaurants (sic) » implantée en Grande-Bretagne n'est là que pour introduire les modules, disponibles sur le CTAN et souvent fournis dans

les distributions récentes, qui proposent des macros pour accomplir ce que nous voulions.

4.1 Le module `verbatim`

Le module `verbatim`[2], dont la documentation est disponible dans le TEXlive 2003, fournit, outre des environnements améliorés pour écrire du texte *verbatim*, un environnement `comment` qui a pour objet de garder caché le texte qu'il contient.

Le `\begin{comment}` et le `\end{comment}` doivent impérativement être seuls sur leur ligne et n'être précédés de rien pas même d'un espace. On écrira :

18 - comment de verbatim

```
Du texte qui sera lisible
\begin{comment}
  Des annotations et des commentaires qui resteront cachés
  dans leur environnement douillet.
\end{comment}
et certainement lu avec grand intérêt.
```

pour obtenir :

Du texte qui sera lisible et certainement lu avec grand intérêt.

Cet environnement fait appel aux commandes `\@bsphack` et `\@esphack` présentées ci-dessus dans l'exemple 17.

Nous pouvons également donner le nom de notre choix à l'environnement `comment` de la manière suivante :

19 - notre comment

```
\let\truc=\comment \let\endtruc=\endcomment
Du texte et du texte
\begin{truc}
  et encore du texte pour voir
\end{truc}
et enfin autre chose.
```

pour obtenir :

Du texte et du texte et enfin autre chose.

L'intérêt que présente cette manœuvre est d'avoir plusieurs niveaux de commentaires et de permettre plus tard de redéfinir l'environnement `truc` pour

2. Rainer Shöpf, Bernd Raichle et Chris Rowley ; version 1.5p, datée 2001-03-12.

faire apparaître les commentaires qui y sont cachés. Mais cela nécessite un `\renewenvironment{truc}`.

4.2 Le module `comment`

Ce module[3] dû à Victor Eijkhout fournit sa documentation sous forme de commentaires dans le fichier `comment.sty`.

Il fournit un environnement `comment` qui doit respecter les contraintes déjà énoncées — voir section 4.1. Mais, contrairement à l'environnement homonyme de `verbatim`, il n'agit pas comme un véritable environnement à la LaTeX : le couple `\begin \end` n'isole pas le commentaire du reste du source et si l'on y place une commande de formatage — quelque chose comme un `\Large` — son effet s'en fait ressentir, lorsque le commentaire est visible dans le document final, dans le texte qui suit, à moins que l'on ait pris la peine de placer quelque barrière autour. C'est cela que je qualifierai de « quasi-environnement » par la suite.

Il fournit aussi la commande `\includecomment` qui permet de définir un pseudo-environnement dont le contenu est systématiquement inclus dans le document et la commande `\excludecomment` qui permet de définir un pseudo-environnement dont le contenu est systématiquement exclu du document, donc considéré comme un commentaire. En voici un exemple.

20 versiona

```
1  \includecomment{versiona}
2  \excludecomment{versionb}
3  \begin{versiona}
4    Ceci apparaîtra dans le document. \bfseries Ah !
5  \end{versiona}
6  Du texte quelconque.
7  \begin{versionb}
8    Mais ceci n'apparaîtra pas et ceci \Large est sans effet
9  \end{versionb}
10 Encore du texte.
```

On obtient alors :

Ceci apparaîtra dans le document. **Ah ! Du texte quelconque. Encore du texte.**

Grâce à la ligne 1, le contenu des environnements `versiona` est montré. La ligne 2 cache le contenu des `versionb`. Il en découle que la commande `\bfseries` de la ligne 4 affecte le texte qui suit ligne 10 mais que le `\Large` de la ligne 8 est sans effet.

3. version 3.6, d'octobre 1999.

Le module fournit, avec la commande `\specialcomment`, le moyen de définir de vrais environnements dont on peut inclure ou exclure le contenu tout en en précisant l'apparence lorsqu'il se retrouve dans le document final. En voici la syntaxe :

21 \specialcomment

```
\specialcomment{Nom}{faire avant}{faire après}
```

Et voici l'exemple fournit par Victor Eijkhout lui-même dans `comment.sty` :

22 un exemple

```
\specialcomment{smalltt}
  {\begingroup\ttfamily\footnotesize}
  {\endgroup}
```

Cela crée un « vrai » environnement grâce au `\begingroup` de la ligne 2 et son petit camarade `\endgroup`. Par défaut, après cette définition, les commentaires embrassés par les `smalltt` ne sont pas cachés. Si l'on veut les exclure du document final, il suffit de placer un `\excludecomment{smalltt}` après la définition.

V. Eijkhout fournit un *truc* dans son commentaire pour définir une commande permettant d'inclure ou d'exclure de petits commentaires.

23 - \maybe

```
\includecomment{cond}
\newcommand{\maybe}[1]{}
\begin{cond}
  \renewcommand{\maybe}[1]{#1}
\end{cond}
```

À la suite de ces déclarations, la commande `\maybe` écrit bêtement son argument. Pour qu'elle le garde caché, remplacer le `in` de `\includecomment{cond}` par un `ex` et le tour sera joué.

Afin de traiter les *environnements* que l'on vient d'évoquer, le module `comment` passe dans un mode *verbatim*, lit ligne par ligne et jette le contenu à moins que le contenu de l'environnement doive être inclus auquel cas il l'écrit dans un fichier dont le nom est donné par `\CommentCutFile` qui vaut `comment.cut` par défaut. On retrouvera donc dans son répertoire de travail un fichier `comment.cut` contenant le dernier morceau contenu dans l'un des environnements de `comment` devant apparaître dans le document final.

4.3 Le module `versions`

Ce module[4] est du à Uwe Lück. Sa documentation est incluse dans le fichier `versions.sty` sous forme d'un commentaire exhaustif. Il n'est pas présent dans la TEXlive 2003 mais sur le CTAN dans `macros/latex/contrib/versions`.

On déclare un environnement et on fixe son comportement avec une seule commande, l'une des trois suivantes :
- `\excludeversion{LaVersion}` qui crée l'environnement `LaVersion` et le définit comme devant cacher le texte qu'il contient ;
- `\includeversion{LaVersion}` qui crée l'environnement `LaVersion` et le définit comme devant montrer le texte qu'il contient ;
- `\markversion{LaVersion}` qui crée l'environnement `LaVersion` et le définit de telle sorte que le texte qu'il contient sera montré mais encadré par les résultats de `\beginmarkversion` et `\endmarkversion`.

Ces trois commandes émettent un message et même un avertissement si l'environnement est déjà défini par ailleurs *mais elles le redéfinissent*. Le code :

24 - entête
```
\documentclass[a4paper,10pt]{article}
\usepackage[latin9]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[frenchb]{babel}
\usepackage{versions}
\begin{document}
\includeversion{VersionA}
\excludeversion{VersionB}
\markversion{VersionC}
\includeversion{otherlanguage}
```

provoquera la trace suivante dans le fichier `log` :

25 - messages
```
*** ‘VersionA’ included. ***
*** ‘VersionB’ excluded. ***
*** ‘VersionC’ included with marks. ***
*** ‘otherlanguage’ included. ***

Package ‘Versions’ Warning: Redefining environment
‘otherlanguage’ on input line 10.
```

4. version 0.5, datée du 2003-10-10.

Par défaut, le module fournit un environnement `comment` qui est défini comme par `\excludeversion{comment}`.

L'auteur met en garde contre le caractère « fragile » de ces environnements. On ne peut les utiliser dans des notes de bas de page par exemple. Mais le module propose la commande `\processifversion` que l'on utilisera comme suit :

26 - \processifversion

```
\processifversion{LaVersion}{texte optionnel}
```

L'apparition du « texte optionnel » est gouvernée par la définition de l'environnement `LaVersion` : présent avec `\includeversion`, il sera caché avec `\excludeversion` et, enfin, présent et marqué avec `\markversion`.

Cette commande est robuste au sens de LaTeX et des arguments mobiles. On pourra donc l'utiliser dans des notes de bas de page, dans des commandes de sectionnements comme `\chapter`, &c.

Lorsque le module est appelé avec l'option `nogroup`, on dispose également de la commande `\includeversionnogroup` qui permet de créer un quasi-environnement, au sens donné plus haut — voir la section 4.2, page 64.

L'option `tracing` permet d'obtenir du module qu'il vérifie la définition des commandes et qu'il fournisse un rapport plus détaillé sur le traitement des environnements du type `Version`. En effet dans ce cas, le module signale les numéros des lignes exclues du document final du fait d'un `\excludeversion`.

Avertissement : le module `versions` est incompatible avec le module — notez l'absence de s final — `version` de Stephen Bellantoni, placé sur CTAN dans `macros/latex/contrib/misc`, qui date de 1990. Ce dernier, créé pour LaTeX 2.09, devrait, à en croire une note de Donald Arseneau datée d'avril 2000, continuer à fonctionner avec LaTeX 2ε mais son utilisation peut entraîner une erreur du type « dépassement de mémoire » si le texte optionnel à exclure est trop important.

4.4 À propos des quasi-environnements

On pourrait se demander pourquoi le module `versions` tient à fournir un moyen de créer des quasi-environnements. En fait, c'est un moyen pratique d'appliquer, de manière optionnelle, certaines commandes fixant l'aspect du texte dans le document final. En voici un exemple jouant sur la couleur. On suppose que l'on a chargé le module `color` ou `xcolor` et que l'on a passé l'option `nogroup` à `versions`.

27 en couleur ou non

```
1 \definecolor{darkblue}{rgb}{0.9,0.05,0.05}
2 \includeversionnogroup{encouleur}
3 %\excludeversion{encouleur}
4 Du texte avant\bgroup
5 \begin{encouleur}\color{darkblue}\end{encouleur}
6 et après.\egroup
```

Tel quel, le code de l'exemple 27, fera que le texte « et après. » sera imprimé en bleu foncé. Il suffira de commenter la ligne 2 et de décommenter la suivante, ligne 3, pour que le texte soit imprimé en noir. Les `\bgroup` et `\egroup`, lignes 4 et 6, sont là pour fixer la partie du code concernée par la commande `\color`. Cet effet n'aurait pas été possible avec un véritable environnement — si l'on avait codé `\includeversion{encouleur}` — puisque l'action de la commande `\color` se serait arrêtée à la sortie de l'environnement, ligne 5. Il faudra, là encore, vérifier que l'on n'a pas introduit d'espace indésirable dans l'une ou l'autre version.

4.5 Le module `optional`

Ce module[5] est du à Donald Arseneau. Il est présent dans la TEXlive 2003, dans le répertoire `texmf/tex/latex/ltxmisc` et sur le CTAN dans `macros/latex/contrib/misc`. Sa documentation est fournie dans `optional.sty` et il est présenté dans le « LaTeX Companion, Second Edition ».

L'idée principale est de sélectionner les textes optionnels à inclure à l'aide d'options passées au module soit dans le `\usepackage` soit, nous y reviendrons, au moment de la compilation.

Deux commandes servent à traiter le texte optionnel : `\opt` et `\optv`. Toutes deux ont la même syntaxe : `\opt{<`*Option*`>}{<`*texte optionnel*`>}` mais `\opt` ne supporte pas de `verbatim` dans son 2e argument, ce que `\optv` fait par contre vaillamment.

D. Arseneau fournit les exemples suivants :

28 optional

```
1 \opt{opta}{Fais ceci si l'option opta est déclarée}
2 \opt{optb}{Fais ceci si l'option optb est déclarée}
3 \opt{optx,opty}{Fais ceci si l'une des options optx ou opty est
4   déclarée}
5 \opt{}{N'imprime jamais ce texte}
```

5. ver 2.2 ; septembre 2001.

```
\opt{opta}{\input{appendices}}
\optv{xam}{Tapez : \verb|[root /]$ rm -r *|.}
```

On voit que les commandes `\opt[v]` acceptent une liste de noms d'options comme premier argument, tout comme on peut passer une liste d'options au module dans `\usepackage`[6].

L'obtention d'une version différente passe par une modification des options passées au module, un peu, note D. Arseneau, comme avec un `\includeonly`. Il y a cependant moyen de procéder autrement.

On peut passer des options par la ligne de commande — avec les systèmes d'exploitation qui savent ce que c'est — au moment du lancement de la compilation :

29 à la compilation

```
latex "\def\UseOption{opta,optb}\input{monfichier}"
```

qui réalisera la même chose que la compilation de `monfichier` dans le préambule duquel on aurait placé

30 préambule

```
\usepackage[opta,optb]{optional}
```

Telle quelle, cette commande est utilisable sur un *nix, pour d'autres systèmes il conviendra d'adapter les caractères de citation.

On peut encore procéder différemment. Si le source contient

31 \AskOption

```
\usepackage{optional}
\newcommand{\ExplainOptions}{man = manuel, check = checklist,
  ref = reference, post = poster.}
\AskOption
```

le module demandera, lors de la compilation, quelles doivent être les options déclarées. Autant `\ExplainOptions`, qui permet d'afficher un message d'explication, que `\AskOption` sont facultatifs. Cette méthode est retenue dès que l'on ne passe aucune option au module dans le source[7]. L'auteur prévient que ce *modus operandi* risque d'être assez rapidement fastidieux.

6. L'auteur déclare dans son commentaire du module : « On permet les listes dans ces deux endroits pour en rendre l'usage plus souple. (Mais en en rendant les définitions plus difficiles, Grrr) ».

7. Ce qui ne manquera pas de poser quelques problèmes si on compile à partir d'un éditeur qui cache l'interaction avec LaTeX comme emacs par exemple.

Avertissements : le texte optionnel argument de `\optv` doit avoir des parenthèses correctement associées sous peine de provoquer quelques désagrément. De plus, si le texte optionnel est trop gros, il faudra avoir recours à un `\opt{opta}{\input{legrosbout}}`, le fichier `legrosbout.tex` contenant le texte optionnel.

5 Avec `docstrip`

Abordons maintenant l'utilisation de `docstrip.tex` et de son langage à balises pour produire à partir d'un source unique plusieurs documents.

`docstrip.tex`[8] est une œuvre collective dont les auteurs sont, par ordre d'apparition dans le fichier lui-même — et aussi par ordre chronologique — Frank Mittelbach, Johannes Braams, Denys Duchier, Marcin Woliński et Mark Wooding. Le dernier copyright est détenu par les auteurs sus-cités et le « LaTeX 3 project ».

C'est un fichier écrit en TeX et non pas en LaTeX aussi son utilisation peut paraître un peu exotique à qui n'a jamais tapé du « bon vieux TeX ». Cependant il définit un système de balises qui permet d'extraire divers documents d'un source unique balisé et, dans l'autre sens, de fabriquer un seul document à partir de plusieurs sources. Ici il faut entendre « document » au sens de « document source LaTeX ». Il restera à compiler les fichiers `.tex` obtenus.

En fait, je triche un peu lorsque je dis « source unique » puisque nous utiliserons en général deux fichiers que l'on nommera souvent sur le modèle `toto.tex` et `toto.ins` mais les extensions n'ont pas de valeur en elles-mêmes pour les manipulations que nous allons faire. Le « vrai » source est `toto.tex` et `toto.ins` est appelé en général *fichier d'installation* qui traduit, imparfaitement car trop spécifiquement, l'anglais *batchfile*.

Je n'aborderai pas toutes les subtilités d'écriture des fichiers `.dtx` que l'on a souvent l'occasion de rencontrer lorsque l'on désire installer un nouveau module ou une nouvelle classe et je ne traiterai pas du tout les fonctionnalités de la classe `ltxdoc` ni du module `doc.sty`. Tout cela va bien au-delà de mon propos.

5.1 Les balises sous les yeux

Commençons par découvrir, peut-être, les balises et leurs utilisations.

Supposons dans un premier temps que notre source `toto.tex` doive donner naissance, après un traitement que nous aborderons plus bas, à trois fichiers nommés

8. version 2.5b datée du 1998-04-28.

respectivement `totoA.tex`, `totoB.tex` et `totoC.tex`.[9] Il nous faudra utiliser au moins trois types de balises que j'identifierai par `VA`, `VB` et enfin `VC`.

5.1.1 Balisons

Regardons un premier exemple que nous compliquerons par la suite :

32 1er exemple de balises

```
\documentclass[a4paper,10pt]{article}
\usepackage{monstyle}
\begin{document}
Pour tout le monde.
%<*VA>
Version A sélectionnée.
%</VA>
%<*VA&VB>
Quand VA et VB sont sélectionnés.
%</VA&VB>
Du texte au milieu pour voir. Sera dans toutes les versions.
%<*VA|VB>
Si VA ou VB est sélectionné mais pas ailleurs.
%</VA|VB>
%<*VA> Ceci ne sera pas repris.
Dans la version A seulement
% Une ligne de commentaire qui disparaît.
 % Une ligne qui restera car il y a un espace en colonne 1.
%% Ceci reste...
%mais pas ceci.
%<*VC>
[ Dans la version C si A est sélectionnée ]
%</VC>
toujours dans A. % Un commentaire qui reste !
%</VA>
%<*!VA>
Partout sauf dans A.
%</!VA>
\end{document}
```

Ce premier usage des balises rappelle celui des environnements de LaTeX mais avec des contraintes bien plus fortes. Une balise commence par `%<` et se termine par `>`.

9. J'espère que ma fantaisie n'a égaré personne !

Le deuxième caractère indique si la balise ouvre `*` ou ferme `/` un *environnement*. Le nom de la balise vient ensuite. On voit une balise ouvrant l'environnement `VA` en ligne 5 à quoi répond la balise fermante de la ligne 7.

On peut imbriquer les environnements, comme en lignes 21 et 23, mais pas les croiser. On retrouve la règle bien connue des utilisateurs de LaTeX, d'ailleurs commune à tous les langages à balises.

On dispose d'un certain nombre d'opérations « logiques » : en ligne 8 on utilise `&` qui indique le « et » ; en ligne 12 ou écrit un « ou » avec `|` ; en ligne 26 on montre la négation, obtenue avec `!`.

Pour les spécialistes, signalons que les expressions booléennes ainsi définies sont évaluées paresseusement de gauche à droite. Pour le laïc cela signifie que `A|B|C` est vrai dès la première variable vraie — et l'évaluation s'arrête là. De même `A&B` est faux dès la première variable fausse. Par ailleurs `A|B&!C` signifie ce que l'on écrirait plus proprement comme suit si l'on pouvait utiliser des parenthèses : `A | (B & !C)`. Quoiqu'il en soit, il sera préférable de créer une balise de plus plutôt que de tenter des constructions trop complexes.

Il existe une deuxième forme de balise qui ne concerne qu'*une* seule ligne en tête de laquelle on la place. Elle prend deux aspects. Le premier `%<+VA>` indique que le reste de la ligne doit être inclus dans le document si `VA` est sélectionné ; la deuxième `%<-VA>` inclut le reste de la ligne si `VA` n'est *pas* sélectionné [10]. Cette deuxième forme admet les mêmes opérateurs logiques que la première.

Enfin, il existe une balise par défaut : l'absence de balise ! Le texte non balisé se retrouve dans tous les fichiers à moins que ce soit une ligne *commençant* par *un seul* % placé en colonne 1. Car un des buts de `docstrip` est bien de faire disparaître toutes les lignes de commentaires du source. Par contre, le caractère % reste lorsqu'il est placé ailleurs. Revoyez l'exemple 32 et le fichier `totoAC.tex` ci-dessous — exemple 35.

Remarque : Le caractère `%` est indispensable au fonctionnement du système. Il fait partie intégrante de la balise. Le cas très spécifique de la balise `<*driver>` des fichiers `.dtx` ne sera pas abordé ici.

5.1.2 Extrayons

Maintenant que notre source est prêt, il paraît légitime de se poser la question suivante : « Comment diable extraire nos documents ? »

10. et non pas, comme j'ai cru un moment, « n'inclut pas si `VA` est sélectionné ».

Pour ce faire, nous devons écrire un fichier `toto.ins` que nous ferons compiler par LaTeX. Cela ne produira pas de fichier `dvi` et de ce fait nous lirons à l'écran « No pages of output. » mais il produira les fichiers attendus.

33 toto.ins

```
1  \input docstrip.tex
2  \keepsilent
3  \nopreamble
4  \nopostamble
5  \askforoverwritefalse
6  \generate{
7    \file{totoA.tex}{\from{toto.tex}{VA}}
8    \file{totoB.tex}{\from{toto.tex}{VB}}
9    \file{totoC.tex}{\from{toto.tex}{VC}}
10   \file{totoAC.tex}{\from{toto.tex}{VA,VC}}
11 }
12 \endbatchfile
```

Tout cela a un bon goût de TeX pur et simple ! En effet, c'est bien du TeX. On voit que l'on charge `docstrip`, en ligne 1, que l'on demande que le processus s'effectue sans donner de nouvelles, ligne 2, ni poser de questions, ligne 5, que les documents produits ne contiendront rien d'autre que ce qui est dans le source c-à-d. ni préambule, ligne 3, ni de « postambule », ligne 4.

L'utilisation du `\generate` est telle qu'elle permet à `docstrip` de créer tous les documents en une seule passe.

Le fichier *doit* être terminé par `\endbatchfile` comme en ligne 12.

On a donc placé dans le même répertoire `toto.tex` et `toto.ins`, on lance `latex toto.ins` et, après quelques instants, on voit apparaître un message qui se termine, à la mise en forme près, par :

34 écran

```
1  Generating file(s) ./totoA.tex ./totoB.tex
2  ./totoC.tex ./totoAC.tex
3  Processing file toto.tex (VA) -> totoA.tex
4                           (VB) -> totoB.tex
5                           (VC) -> totoC.tex
6                           (VA,VC) -> totoAC.tex
7  Lines  processed: 34
8  Comments removed: 2
9  Comments  passed: 5
10 Codelines passed: 14
```

```
)
No pages of output.
Transcript written on toto.log.
```

Regardons tout de suite le fichier `totoAC.tex` :

35 totoAC.tex

```
\documentclass[a4paper,10pt]{article}
\usepackage{monstyle}
\begin{document}
Pour tout le monde.
Version A sélectionnée.
Du texte au milieu pour voir. Sera dans toutes les versions.
Si VA ou VB est sélectionné mais pas ailleurs.
Dans la version A seulement
 % Une ligne qui restera car il y a un espace en colonne 1.
%% Ceci reste...
[ Dans la version C si A est sélectionnée ]
toujours dans A. % Un commentaire qui reste !
\end{document}
```

On trouvera à la fin de cet article le contenu des autres fichiers créés par cette opération.

5.1.3 Quelques remarques sur le préambule

Par défaut, `docstrip` écrit en tête du fichier créé un préambule signalant comment ledit fichier fut obtenu. Ces indications ne sont pas toujours d'un intérêt extrême pour ce que nous voulons faire. Le « postambule » contient lui la seule commande `\endinput`.

La documentation du module aborde le problème de la modification de ces deux morceaux de fichier mais je n'ai pas trouvé que, pour ce qui nous occupe ici, le jeu en valait la chandelle. J'ai préféré couper court et me débarrasser de ces deux bouts de texte.

Il est un autre problème : avec emacs et son mode auctex, j'ai pris l'habitude d'insérer deux lignes d'entête dans mes fichiers et d'y laisser s'inscrire en fin les indications nécessaires au fonctionnement d'auctex. Pour la fin, en fait, pas d'ennui à craindre puisque les lignes commencent par `%%%`. Mais l'entête est le suivant :

36 - avec emacs

```
% -*- mode: LaTeX; coding: iso-8859-15 -*-
% Time-stamp: <2004-06-19 21:17:06 yvon>
```

Bien entendu, je pourrais m'arranger pour que ces lignes commencent par deux signes % et emacs n'y trouverait rien à redire. Mais je peux malgré tout forcer `docstrip` à les incorporer *verbatim* dans les fichiers extraits. Il suffit d'utiliser ceci :

37 - recopier verbatim

```
%<<ENTETE
% -*- mode: LaTeX; coding: iso-8859-15 -*-
% Time-stamp: <2004-06-19 21:17:06 yvon>
%ENTETE
```

Le mot « ENTETE » n'a aucune signification particulière, c'est la manière dont il est placé qui impose à `docstrip` de recopier *verbatim* toutes les lignes situées entre le `%<<ENTETE` et `%ENTETE`. Cela rappellera à d'aucuns les « *here-documents* » de Perl par exemple.

5.2 Un exemple concret

Considérons le petit scénario suivant : j'écris un document composé d'exercices suivis de leurs solutions. J'ai créé un module `monstyle` qui charge tous les modules nécessaires, définit un certain nombre de commandes personnelles et également deux environnements : `exercice` et `correction`.

Dans un premier temps, j'écris un exercice puis sa correction, puis un autre exercice suivi de sa correction, &c mais je me dis que j'aimerais bien avoir plusieurs versions de ce travail. La première, destinée à être placée au vu de tous sur mon site personnel, ne doit comprendre que les énoncés. Pour ce qui est de la version corrigée, j'en ferais bien deux variantes : la première reprend tout bonnement la présentation du source avec son alternance d'énoncés et de solutions mais en chargeant un autre module qui assurerait que l'on change de page à chaque nouvel exercice. Cette première version, avec corrigés, aurait vocation à être distribuée feuille à feuille. La deuxième présenterait d'abord tous les exercices puis, après un texte de liaison, tous les corrigés. Chacun des exercices ayant son propre corrigé, je pourrais, quitte à charger un autre module de mise en forme des environnements `exercice` et `correction`, obtenir une numérotation parallèle qui conviendra tout à fait à mes souhaits.

Le problème n'est donc plus que d'extraire ce qu'il faut de mon source correctement balisé et, aussi, de concaténer des fichiers pour placer toutes les corrections après tous les exercices dans un même document LaTeX.

Je propose à ce petit problème la solution, double, que voici :

38 sources.tex

```
%<*!trans>
%<*!sansdeb>
%<<ENTETE
%% -*- mode: LaTeX; coding: iso-8859-15 -*-
%% Time-stamp: <2004-06-19 20:42:36 yvon>
%ENTETE
\documentclass[a4paper,10pt]{maclasse}
\usepackage{commun}
%</!sansdeb>
%<+dtch>\usepackage{mesexoscorr}
%<+sans>\usepackage{mesexosseuls}
%<+avec>\usepackage{mesexos}
%<*!sansdeb>
\begin{document}
%</!sansdeb>
%<*enon>
\begin{exercice}
  L'énoncé du premier exercice.
\end{exercice}
%</enon>
%<*corr>
\begin{correction}
  La correction du premier exercice.
\end{correction}
%</corr>
%<*enon>
\begin{exercice}
  L'énoncé du deuxième exercice.
\end{exercice}
%</enon>
%<*corr>
\begin{correction}
  La correction du deuxième exercice.
\end{correction}
%</corr>
%</!trans>
%<*trans>
Ici se place le code qui fera la transition entre
la série des énoncés et la série des corrections.
%</trans>
%<*!trans>
```

```
%<*!sansfin>
Code de fin pour les documents complets.
\tableofcontents \printindex
\end{document}
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
%</!sansfin>
%</!trans>
```

39 sources.ins

```
\input docstrip.tex
\keepsilent
\nopreamble
\nopostamble
\askforoverwritefalse
\generate{
  \file{exocorr.tex}{\from{sources.tex}{enon,corr,avec}}
  \file{exoseul.tex}{\from{sources.tex}{enon,sans}}
  \file{sanscor.ltx}{\from{sources.tex}{enon,dtch,sansfin}}
  \file{detache.tex}{\from{sources.tex}{sansdeb,corr}}
  \file{transit.ltx}{\from{sources.tex}{trans}}
}
\generate{
  \file{cordtch.tex}{
    \from{sanscor.ltx}{}
    \from{transit.ltx}{}
    \from{detache.tex}{}
  }}
\endbatchfile
```

Annexes

Le tableau suivant ne reprend pas toujours toutes les commandes disponibles dans un module donné mais uniquement celles évoquées ci-dessus dans le but de traiter des commentaires ou du texte optionnel.

ifthen	
`\ifthenelse`	`\ifthenelse{<`*test*`>}` `{<`*action si oui*`>}{<`*action si non*`>}`
`\and`	`<`*booléen 1*`>` `\and` `<`*booléen 2*`>`
`\or`	`<`*booléen 1*`>` `\or` `<`*booléen 2*`>`
`\not`	`\not` `<`*booléen*`>`
`\(` et `\)`	`\(` `<`*booléen*`>` `\)`
`\equal`	`\equal{<`*chaîne 1*`>}{<`*chaîne 2*`>}`
`\newboolean`	`\newboolean{<`*Booléen*`>}`
`\setboolean`	`\setboolean{<`*Booléen*`>}{<`*true* ou *false*`>}`
`\boolean`	`\boolean{<`*Booléen*`>}`
comment	
`comment`	quasi-environnement voir page 64
`versiona`	quasi-environnement
`versionb`	quasi-environnement
`smalltt`	exemple construit avec `\specialcomment`
`\includecomment`	`\includecomment{<`*nom de quasi-env.*`>}`
`\excludecomment`	`\excludecomment{<`*nom de quasi-env.*`>}`
`\specialcomment`	`\specialcomment{<`*non d'environnement*`>}` `{<`*faire avant*`>}{<`*faire après*`>}`
`\CommentCutFile`	produit le nom du fichier de travail de `comment`, par défaut : `comment.cut`
`\maybe`	exemple de commande
versions	
`comment`	texte exclu par défaut
`\includeversion`	`\includeversion{<`*version*`>}`

... versions suite

`\excludeversion`	`\excludeversion{<`*version*`>}`
`\markversion`	`\markversion{<`*version*`>}`
`\beginmarkversion`	utilisé automatiquement
`\endmarkversion`	*idem*
`\processifversion`	`\processifversion{<`*version*`>}{<`*optionnel*`>}`
	`optional`
options	définies par l'utilisateur
`\opt`	`\opt{<`*option*`>}{<`*optionnel*`>}`
`\optv`	`\optv{<`*option*`>}{<`*verbatim optionnel*`>}`
`\UseOption`	pour utilisation dans la ligne de commande
`\ExplainOptions`	`\ExplainOptions{<`*message explicatif*`>}`
`\AskOption`	`\AskOption`
	`verbatim`
`comment`	environnement voir page 63
`\comment`	pour définir son propre environnement
`\endcomment`	*idem*, voir exemple 19
	`docstrip`
commandes pour le « fichier `ins` »	
`\nopreambule`	`\nopreambule`
`\nopostambule`	`\nopostambule`
`\keepsilent`	`\keepsilent`
`\askforoverwritefalse`	`\askforoverwritefalse`
`\endbatchfile`	`\endbatchfile`
`\generate`	voir l'exemple 33
`\file`	*idem*
`\from`	*idem*

Les fichiers créés dans l'exemple 33

40 - totoA

```
\documentclass[a4paper,10pt]{article}
\usepackage{monstyle}
\begin{document}
Pour tout le monde.
Version A sélectionnée.
Du texte au milieu pour voir. Sera dans toutes les versions.
Si VA ou VB est sélectionné mais pas ailleurs.
Dans la version A seulement
 % Une ligne qui restera car il y a un espace en colonne 1.
%% Ceci reste...
toujours dans A. % Un commentaire qui reste !
\end{document}
```

41 - totoB

```
\documentclass[a4paper,10pt]{article}
\usepackage{monstyle}
\begin{document}
Pour tout le monde.
Du texte au milieu pour voir. Sera dans toutes les versions.
Si VA ou VB est sélectionné mais pas ailleurs.
Partout sauf dans A.
\end{document}
```

42 - totoC

```
\documentclass[a4paper,10pt]{article}
\usepackage{monstyle}
\begin{document}
Pour tout le monde.
Du texte au milieu pour voir. Sera dans toutes les versions.
Partout sauf dans A.
\end{document}
```

Index

\(, 58
\), 58
\AskOption, 69
\CommentCutFile, 65
\ExplainOptions, 69
\MonComfalse, 57
\MonComtrue, 57
\UseOption, 69
\and, 58
\askforoverwritefalse, 73
\beginmarkversion, 66
\boolean, 58
\@bsphack, 62, 63
\comment, 63
\else, 56
\endbatchfile, 73
\endcomment, 63
\endmarkversion, 66
\equal, 58
\@esphack, 62, 63
\excludecomment, 64
\excludeversion, 66
\fi, 56
\from, 73
\generate, 73
\ifMACHIN, 56
\iffalse, 56
\ifthenelse, 58
\includecomment, 64
\includeversion, 66
\keepsilent, 73
\markversion, 66
\maybe, 65
\newif, 57
\nopostamble, 73
\nopreamble, 73
\not, 58
\opt, 68
\optv, 68, 70
\or, 58
\processifversion, 67
\setboolean, 58
\specialcomment, 65

ARSENEAU, Donald, 67, 68
BELLANTONI, Stephen, 67
BRAAMS, Johannes, 70
CARLISLE, David, 58
DUCHIER, Denys, 70
EIJKHOUT, Victor, 64, 65
LAMPORT, Leslie, 58
LÜCK, Uwe, 66
MITTELBACH, Frank, 70
WOLIŃSKI, Marcin, 70
WOODING, Mark, 70

balises, 70–72
    verbatim, 75
batchfile, 70

comment – ENV, 63, 64, 67
comment – MOD, 64
comment.cut, 65
commentaires, 54

docstrip – MOD, 70
document final, 54

Environnements
    comment, 63, 64, 67
    smalltt, 65

fichier d'installation, 70
fichier .ins, 70

ifthen – MOD, 58

ligne de commande, 69

Modules
    `comment`, 64
    `docstrip`, 70
    `ifthen`, 58
    `optional`, 68
    `verbatim`, 63
    `version`, 67
    `versions`, 66

`nogroup` – *option de* `versions`, 67

`opta` – *option de* `optional`, 69
`optb` – *option de* `optional`, 69
`optional` – MOD, 68

quasi-environnement, 64, 67

`smalltt` – ENV, 65

texte optionnel, 54
`tracing` – *option de* `versions`, 67

`verbatim` – MOD, 63
`version` – MOD, 67
`versions` – MOD, 66