Astuce 8 - \def vs \newcommand

Une activité essentielle dans l'usage de (La)TeX est la définition de macros. Pour cela, LaTeX propose des méthodes qui lui sont propres (\newcommand, \renewcommand, etc.) en plus des méthodes de TeX qui restent disponibles. Comme souvent, on a intérêt sauf raison particulière à préférer les méthodes de LaTeX pour définir les commandes.

La première raison est que LaTeX opère beaucoup d'opérations de contrôle pour vous empêcher de tout casser par maladresse. En l'occurrence, le danger est de redéfinir sans le savoir une commande qui existe déjà (vous ne connaissez sans doute pas par coeur la liste des 300 primitives de TeX et encore moins celle des commandes LaTeX). S'il s'agit d'une commande ayant un rôle essentiel (comme \par ou \output), les conséquences peuvent être catastrophique et difficiles à comprendre.

Pour cela, LaTeX distingue la définition d'une nouvelle commande qui se fait avec \newcommand de la redéfinition qui se fait avec \renewcommand. Par ailleurs, il faut savoir qu'en LaTeX il est dangereux de définir des commandes dont le nom commence par \end car elles sont utilisée en interne par les environnements. Là aussi, \newcommand se charge de la vérification et vous évite des problèmes.

Par ailleurs, une raison un peu plus personnelle de préférer \newcommand est sa syntaxe plus simple pour la spécification du nombre d'arguments, et en un sens plus puissante avec la possibilité de rendre optionnel le premier argument. Concernant les arguments, \def offre d'autres possibilités, comme les arguments délimités, dont on a rarement besoin sauf pour des techniques un peu avancées.

Dans les rares cas où on a vraiment besoin de \def (ou d'autres commandes TeX comme \let ou \edef), un usage prudent est de faire précéder la définition de la macro (par exemple \def\macro) par un \newcommand\macro{} pour vérifier qu'on est bien autorisé à définir \macro.

Bien nommer ses macros et environnements

La première chose chose à savoir pour nommer ses macros où environnements est quels sont les noms autorisés. Pour les macros, ils sont de deux types : les caractères de contrôle et les mots de contrôle. Dans un caractère de contrôle (p. ex. \!), la contre-oblique est suivie d'un unique caractère qui n'est pas une lettre. Dans un mot de contrôle, la contre-oblique est suivie d'une suite de lettres et le nom de la commande se termine au premier caractère non-lettre (typiquement un espace ou une accolade).

TeX reconnaît les 52 caractères auxquels vous pensez comme des lettres. En particulier, A et a sont différents, et é, @ ou 1 ne sont pas des lettres. On peut parfois demander à TeX de changer sa notion de lettre : voir par exemple le cas de \makeatletter.

Pour les noms d'environnements, c'est plus simple : les caractères autorisés sont les lettres et le caractère * et ce, quelle que soit la longueur. Vous pouvez essayer d'utiliser d'autres caractères comme des espaces, cela marchera sans doute. Cependant, il n'est écrit nulle part dans le manuel que ça doit marcher, donc il n'est pas certain que cela marchera encore à l'avenir et je vous conseille de ne pas le faire.

Le deuxième élément à prendre en compte est le sens du nom. On vous encourage à ne pas le choisir trop court : votre source gagnera en lisibilité, et c'est sans doute plus important que d'économiser quelques frappes de touches. Par ailleurs, choisissez toujours un nom qui se rapporte au sens de la commande et pas à sa mise en forme (par exemple, \lebesgue pour la mesure de Lebesgue, indépendamment du fait que vous la notiez lambda ou autre chose).

Définir des variantes étoilées

Vous l'avez sans doute remarqué, beaucoup d'environnements ou commandes standard de

LaTeX existent sous une variante étoilée. Vous pouvez avoir envie de reprendre cette idée quand vous définissez vos commandes et arguments, et j'approuve cette envie. Voyons comment la mettre en oeuvre.

L'étoile est un caractère autorisé dans les noms d'environnements. Il vous suffit donc de faire \newenvironment{myenv} et \newenvironment {myenv*} avec les définitions souhaitées.

Pour les commandes, c'est plus compliqué car l'étoile ne peut pas faire partie du nom de la commande. Il y aura donc une commande, qui devra être capable de regarder si elle est ou non suivie d'une étoile et d'adapter son comportement en conséquence. Pour des raisons techniques, cette commande ne pourra pas accepter d'argument, mais pourra faire appel à des commandes qui en acceptent. Par ailleurs, on utilise une commande interne du noyau LaTeX et vous pouvez vous reportez à « \makeatletter » pour comprendre pourquoi cela implique son usage.

```
\newcommand*\mycommandstared[1]{ciel #1 étoilé}
\newcommand*\mycommandunstared[1]{ciel #1 non étoilé}
\makeatletter
\newcommand\mycommand{\@ifstar
{\mycommandstared}{\mycommandunstared}}
\makeatother
```

Dans cet exemple (stupide), vous pouvez alors utiliser \mycommand comme une commande avec un argument obligatoire et admettant une variante étoilée. Ainsi, \mycommand{bleu} composera « ciel bleu non étoilé » tandis que \mycommand* {nocturne} composera « ciel nocturne étoilé ».

Gérer astucieusement ses arguments

Dans cette astuce, on suppose que vous connaissez déjà la syntaxe de \newcommand. En particulier, vous savez sans doute qu'on ne peut définir (au plus) qu'un argument optionnel et que celui-ci est nécessairement le premier. L'objet de la présente astuce est de contourner, dans certains cas, cette limitation.

Commençons par une remarque assez fondamentale sur la façon dont TeX gère les macros. D'abord, il faut bien comprendre qu'une macro n'est pas une fonction : il ne s'agit pas d'exécuter à part le code de la macro et de renvoyer un résultat, mais juste de remplacer le nom de la macro par son texte de définition. En particulier, TeX n'opère aucun contrôle sur le texte de définition d'un macro, qui peut contenir des macros non définies ou n'ayant pas le bon nombre d'arguments.

On peut alors méditer sur le cas d'une commande définie comme \newcommand\latin{\textit}. Techniquement, elle semble être une macro sans

argument. En pratique, l'utilisateur écrira \latin{id est} comme si la commande admettait un argument : celui-ci sera en fait passé à \textit. Vous êtes maintenant prêts à comprendre l'exemple suivant.

```
\newcommand*\xvec[1][0]{x_{#1},\ldots,\xvecint}
\newcommand*\xvecint[1][n]{x_{#1}}

\xvec       % donne x_0,\ldots,x_n
\xvec[1]    % donne x_1,\ldots,x_n
\xvec[1][m] % donne x_1,\ldots,x_m
\xvec[m]    % donne x_m,\ldots,x_n (attention)
```

Deux remarques. Si le dernier argument est optionnel et que vous ne l'utilisez pas, votre commande va manger les espaces qui la suivent, soyez prudent hors du mode mathématique. Pour des traitement plus compliqués, vous devrez utiliser des définitions emboîtées pour faire circuler les arguments entre vos différentes macros internes.

Être ou ne pas être une lettre : le cas de @

Vous savez qu'en temps normal, si l'on écrit, par exemple

```
\today@midnight,
```

il ne s'agit pas d'un nom de commande seul, mais de la commande \today suivie des caractères @,

m, i, etc.

Si pourtant vous lisez les sources d'un package, d'un fichier de classe, voire du noyau LaTeX, vous constaterez que beaucoup de noms de commandes y contiennent au moins un @. Il s'agit d'un convention standard de LateX (déjà utilisée par le format plain d'ailleurs) qui sert à « protéger » certaines commandes en les rendant inaccessible à l'utilisateur en temps normal.

Ceci repose que le fait qu'on peut demander à TeX de considérer @ (ou tout autre caractère) comme une lettre si on le désire. Par défaut avec LaTeX, @ est une lettre à l'intérieur des fichiers de style (packages) ou de classe, et n'est pas une lettre dans les documents. Pour certains cas, vous pouvez vouloir utiliser des commandes internes. Il faut alors auparavant dire à TeX de considérer @ comme une lettre avec la commande explicite \makeatletter, et ne pas oublier ensuite de rétablir l'ordre naturel des choses par la commande \makeatother.

Étoiler ses \newcommand

Une astuce qui ne changera rien à vos documents, mais rendra votre code plus robuste en vous permettant de mieux localiser les erreurs. Une des erreurs les plus courantes en tapant un document LaTeX est de mal équilibrer les accolades : en

particulier d'oublier l'accolade fermante à la fin de l'argument d'une macro.

Dans un cas pareil, si rien n'est prévu, TeX lira votre fichier jusqu'à la fin avant de comprendre qu'il y a une erreur et ne pourra rien faire. Pour éviter cela, Knuth (le créateur de TeX), a prévu de distinguer deux types de macros, courtes ou longues, selon que leurs arguments peuvent ou non contenir un changement de paragraphe. Pour une raison qui m'échappe, par défaut, \newcommand définit des macros longues.

En utilisant \newcommand* pour définir votre commande, vous en faites une macro courte, ce qui est largement préférable dans la plupart des cas : ainsi, les erreurs dues à un éventuel oubli d'accolade fermante seront circonscrites à un paragraphe. En cas de problème, TeX s'en rendra compte à la fin du paragraphe (et non du document), il pourra vous signaler facilement l'emplacement de l'erreur et composer le reste du document comme si de rien n'était.

Prenez donc l'habitude d'utiliser la plupart du temps les versions étoilés de \newcommand et \renewcommand (sauf bien sûr dans des cas particuliers), cela vous épargnera quelques désagréments.

Commenter ses fins de ligne

Une autre astuce un peu technique qui peut vous épargner bien des désagréments. Comme vous le savez peut-être, quand TeX lit votre document, il traduit chaque fin de ligne en espace. La plupart du temps (quand vous saisissez le texte d'un paragraphe sur plusieurs lignes par exemple), c'est exactement le comportement qu'il faut pour vous simplifier la vie.

Il y a d'autres circonstances où ce comportement peut être gênant. Notamment, quand vous définissez une macro un peu longue, ou que vous faites quelque chose de compliqué, vous pouvez avoir envie de présenter votre code de façon aérée sur plusieurs lignes (et indenté) pour le rendre plus lisible, comme dans l'exemple suivant.

```
\def\@protected@testopt#1{%
\ifx\protect\@typeset@protect
\expandafter\@testopt
\else
\@x@protect#1%
\fi}
```

Dans ces circonstances, vous ne voulez pas que les retours à la ligne purement liés à l'esthétique de votre code introduisent des espaces dans votre document quand votre macro sera utilisée au milieu du texte. Pour cela, prenez l'habitude d'insérer un caractère de commentaire « % » à la fin de la ligne (ce qu'on appelle souvent «

commenter ses fins de ligne »).

Vous n'êtes pas obligé de commenter les fins de ligne qui suivent un mot de contrôle car, vous le savez (cf noms des macros), les espaces sont ignorés après un mot de contrôle. Vous n'avez pas non plus besoin de commenter la fin de ligne après un chiffre qui n'est pas une spécification d'argument, car, je l'ai déjà évoqué, il est prudent de faire suivre les nombres (sauf spécification d'argument) d'un espace qui sert à les terminer, et est donc avalé par TeX comme l'espace qui termine un mot de contrôle.

Pour vous convaincre que les fins de lignes introduisent bien des espaces que vous de désirez peut-être pas, et qu'on les évite facilement en commentant ses fins de lignes, on propose l'exemple suivant :

```
\fbox{
Du texte.
}
\quad
\fbox{%
Du texte.%
}
```

Prenez donc l'habitude de systématiquement commenter les fins de lignes qui le nécessitent quand vous écrivez des macros ou que vous présentez du code sur plusieurs lignes.