L'extension xargs

Manuel Pégourié-Gonnard
mpg@elzevir.fr

v1.1 (2008/03/22)

1 Introduction

LaTeX 2ε permet de définir facilement des commandes ayant un argument optionel. Cependant, il y a deux restrictions : il peut y avoir au plus un argument optionel, et ce doit être le premier. L'extension xargs fournit des versions étendues de `\newcommand` et de ses analogues standard, qui ne présentent pas ces restrictions : il est désormais facile de définir des commandes avec plusieurs arguments optionnels, placés où l'on veut, par une syntaxe agréable du style ⟨*clé*⟩=⟨*valeur*⟩. Par exemple, voici comment définir une commande avec deux arguments optionnels.

```
\newcommandx*\coord[3][1=1, 3=n]{(#2_{#1},\ldots,#2_{#3})}
```

`$\coord{x}$`	$(x_1,\ldots,x_n)$
`$\coord[0]{y}$`	$(y_0,\ldots,y_n)$
`$\coord{z}[m]$`	$(z_1,\ldots,z_m)$
`$\coord[0]{t}[m]$`	$(t_0,\ldots,t_m)$

2 Usage

2.1 Les bases

L'extension xargs fournit des analogues de toutes les macros de LaTeX 2ε relatives à la définition de macros. Les macros de xargs ont le même nom que leur analogue standard, mais avec un `x` supplémentaire à la fin. En voici la liste complète.

`\newcommandx`	`\renewcommandx`
`\newenvironmentx`	`\renewenvironmentx`
`\providecommandx`	`\DeclareRobustCommandx`
`\CheckCommandx`	

Si vous ne connaissez pas toutes ces commandes, ne vous inquiétez pas. Vous pouvez utiliser seulement les analogues des macros que vous connaissez ; ou apprendre les autres par exemple dans le livre de Lamport, le LaTeX Companion, ou tout autre manuel sur LaTeX 2ε.

Ces commandes partageant toutes la même syntaxe, je parlerai seulement de `\newcommandx` par la suite, mais gardez à l'esprit que les explications sont valables pour les autres commandes. (Bien sûr, pour les environnements, il y a un dernier argument en plus, pour la définition de fin.) Voici la syntaxe complète de `\newcommand`.

`\newcommandx`⟨∗⟩`{`⟨*commande*⟩`}[`⟨*nombre*⟩`][`⟨*liste*⟩`]{`⟨*définition*⟩`}`

Rappelons brièvement tout ce qui est commun avec la syntaxe usuelle, à savoir tout sauf ⟨*liste*⟩. Si une ∗ est présente, elle signifie que la macro crée est *courte* au sens de TEX, c'est-à-dire que ses arguments ne peuvent pas contenir de saut de paragraphe (`\par` ou ligne vide). La ⟨*commande*⟩ est n'importe quelle séquence de contrôle, que vous pouvez ou non entourer d'accolades suivant vos goûts. Le ⟨*nombre*⟩ définit le nombre total d'argument de la ⟨*commande*⟩, c'est un entier compris entre 0 et 9. La ⟨*définition*⟩ est un texte équilibré en accolades, et où chaque caractère `#` est suivi soit d'un chiffre représentant un des arguments, soit d'un autre caractère `#`. Les arguments ⟨*nombre*⟩ et ⟨*liste*⟩ sont optionnels.

La partie intéressante maintenant. La ⟨*liste*⟩ est une... liste (!) d'éléments de la forme ⟨*chiffre*⟩=⟨*valeur*⟩, séparés par des virgules. Le ⟨*chiffre*⟩ doit être un entier compris entre 1 et le nombre d'arguments, donné par ⟨*nombre*⟩. La ⟨*valeur*⟩ est n'importe quelle texte équilibré en accolades. Il peut être vide si vous le souhaitez : le signe égal qui le précède peut alors être omis. Tous les arguments dont le numéro figure en tant que ⟨*chiffre*⟩ dans la ⟨*liste*⟩ seront optionnels, avec pour valeur par défaut celle donnée par la ⟨*valeur*⟩ correspondante.

Quelques remarques supplémentaires sur la syntaxe de la ⟨*liste*⟩, que vous pouvez sauter si vous êtes familiers avec la syntaxe fournie par xkeyval. Vu que les éléments sont séparés par des virgules, si une ⟨*valeur*⟩ doit contenir une virgule, il faut entourer la valeur par des accolades pour protéger la virgule. (Cette précaution est également indispensable si la ⟨*valeur*⟩ contient une accolade fermante.) Ne vous inquiétez pas, cette parie d'accolade sera retirée ultérieurement. D'ailleurs, jusqu'à trois paires d'accolades seront retirées ainsi, et si vous voulez vraiment que votre valeur reste entourée d'accolades, il vous faudra écrire quelque chose comme `1={{{{\large blabla}}}}`.

C'est tout pour les bases : vous en savez maintenant assez pour utiliser xargs, et vous pouvez laisser la fin de la documentation pour plus tard si vous voulez. Si, au contraire, vous vous demandez ce qui se passe avec plusieurs arguments optionnels à la suite, que vous voulez effectuer des définitions globales, ou que vous aves besoin de connaître les limites précises de xargs, vous pouvez lire les sections suivantes.

2.2 La clé `usedefault`

Voyons donc se qui se passe quand plusieurs arguments optionnels se suivent. Le comportement par défaut est calqué sur celui de commandes LATEX comme `\makebox` et `\parbox` : on ne peut spécifier une valeur pour le troisième argument optionnel que si on l'a fait pour les deux premiers. Par exemple, dans l'exemple du début, remarquez comment j'avais pris soin de placer l'argument obligatoire au milieu pour séparer les arguments optionnels.

Cependant, ce n'est pas très pratique, et on aimerait pouvoir choisir l'ordre des arguments plutôt selon leur sens, sans se poser de questions. Bien, la clé `usedefault` est justement là pour ça : incluez-la dans la ⟨*liste*⟩, et vous pouvez désormais utiliser `[]` pour sauter un argument optionnel, en utilisant sa valeur par défaut.

```
\newcommandx*\coord[3][2=1,3=n,usedefault]{(#2_{#1},\ldots,#2_{#3})}
```

`$\coord{x}$`	$(1_x, \ldots, 1_n)$
`$\coord{y}[0]$`	$(0_y, \ldots, 0_n)$
`$\coord{z}[][m]$`	$(1_z, \ldots, 1_m)$
`$\coord{t}[0][m]$`	$(0_t, \ldots, 0_m)$

Bien sûr, sur cet exemple simple, c'est surtout une histoire de goût, mais parfois `usedefault` peut vous épargner pas mal de frappe inutile, car la valeur par défaut d'un argument est parfois longue, et qu'on a pas toujours assez d'arguments obligatoires pour séparer les arguments optionnels comme plus haut.

Cette utilisation simple de `usedefault` présente un inconvénient : vous ne pouvez plus spécifier la valeur vide pour un argument optionnel. En fait, il faut nécessairement une valeur spéciale de l'argument pour dire « utiliser la valeur par défaut », mais ce n'est pas nécessairement la chaîne vide : en fait, on peut choisir cette valeur en disant `usedefault=`⟨*valeur*⟩. L'exemple suivant ne sert à rien, à part illustrer ceci.

```
\newcommandx*\test[2][1=A, 2=B, usedefault=@]{(#1,#2)}
```

`\test[b]`	(b,B)
`\test[][b]`	(,b)
`\test[@][b]`	(A,b)

2.3 Ajouter un préfixe

La commande `\newcommand` standard permet de définir au choix des macros « longues » (dont les arguments peuvent contenir des `\par`) ou « courtes » (les `\par` ou lignes vides sont interdits dans les arguments) au moyen de l'étoile optionnelle. C'est une partie de ce que TEX appelle un préfixe, plus précisément la composante `\long`. Les autres composantes d'un préfixe peuvent être `\global`, `\outer`, ou (avec ε-TEX) `\protected`. Il n'y a pas de moyen d'utiliser ces composantes avec `\newcommand`, bien que, par exemple, `\global` puisse être utile pour qu'une définition faite à l'intérieur d'un groupe (comme un environnement) ne soit pas « oubliée » à la fin. (Pour des détails sur les autres, voir le TEXbook ou le manuel d'ε-TEX.)

Avec xargs, vous pouvez utiliser la clé `addprefix`, *sauf* pour la composante `\outer`, qui n'est pas et ne sera pas supportée (et, à ma connaissance n'est jamais utilisée dans LATEX 2_ε). Remarquons que cette clé *ajoute* une préfixe à celui en cours, elle n'écrase rien. Au début, le préfixe par défaut est `\long`, ou vide si une étoile a été utilisée. Par exemple, les deux instructions suivantes ont exactement le même effet.

```
\newcommandx*\truc[0][addprefix=\global, addprefix=\long,
  addprefix=\protected]{machin}
\newcommandx\truc[0][addprefix=\global\protected]{machin}
```

En passant, les macros avec au moins un argument optionnel sont définies de façon robuste au sens que LATEX 2_ε donne à ce mot, je ne sais donc pas si le préfixe `\protected` est très utile. Je pense que la possibilité d'effectuer des définitions globales est l'usage principale de la clé `usedefault`.

2.4 Compatibilité et limitations connues

La mauvaise nouvelle (les limitations) en premier. Il y en a essentiellement une : on ne peut pas utiliser dans la ⟨*liste*⟩ certaines choses, qui ne sont pas gérées correctement par xkeyval. Précisément, il s'agit des signes `#` (les lexèmes de `\catcode` 6) et des lexèmes `\par`. Aussi, aucune composante de la ⟨*liste*⟩ ne doit avoir l'air mal équilibrée en `\if`s aux yeux de TEX. Seule la première de ces limitations est partagée pas le `\newcommand` standard, qui n'accepte pas non plus de `#` dans les valeurs par défaut. Autrement, vous pouvez utiliser ce que vous voulez, où vous voulez (autant que je sache).

Maintenant les bonnes nouvelles. J'ai pris grand soin que les macros définies avec xargs ressemblent autant que possibles à celles définies avec les commandes standard de LaTeX. En fait, quand on demande à `\newcommandx` d'effectuer une définition que `\newcommand` aurait pu faire, la commande sera définie exactement comme si on avait utilisé ce dernier. Plus précisément, le code suivant (et les tests similaires) ne renvoie pas d'avertissement.

```
\newcommandx\truc[2][1=default]{def-truc}
\CheckCommand\truc[2][default]{def-truc}
\newcommand\chose{def-chose}
\CheckCommandx*\chose[0][addprefix=\long]{def-chose}
```

De plus, il y a seulement trois points (à ma connaissance) sur lesquels les commandes d'xargs diffèrent de celles de LaTeX. Le premier a déjà été évoqué, c'est la limitation sur le ⟨*liste*⟩ due à xkeyval. Les deuxièmes et troisièmes points, par contre, sont censés être positifs. Le deuxième est donc, que j'ai essayé d'éviter de reproduire dans `\CheckCommandx` deux problèmes [1] dont souffre l'implémentation actuelle de `\CheckCommand` dans LaTeX.

Le dernier point concerne la gestion des espaces, lors de la recherche du prochain caractère, pour déterminer s'il s'agit ou non d'un crocher carré, quand on teste la présence d'un argument optionnel. Pour ceci, je n'utilise ni la version du noyau, ni celle d'amsmath, de `\@ifnextchar`, mais la mienne, qui a le comportement suivant : elle avale les espaces jusqu'à trouver le prochain caractère, puis les restitue si ce n'était pas le début d'un argument optionnel. Je ne suis plus tout à fait sûr que ce soit la bonne façon de faire, et il est probable que je ferai une option à ce sujet dans une prochaine version, afin que l'utilisateur puisse choisir son comportement préféré.

Vous savez maintenant absolument tout ce qu'il y a à savoir sur l'utilisation de xargs. Si vous souhaitez vous pencher sur son implémentation, il vous faudra lire les commentaires en anglais car je n'ai pas eu le courage de commenter mon code en deux langues.

C'est tout pour cette fois !
Amusez-vous bien avec LaTeX !

1. http://www.latex-project.org/cgi-bin/ltxbugs2html?pr=latex/3971