L'extension xargs

Manuel Pégourié-Gonnard
mpg@math.jussieu.fr

v1.0 (2007/10/20)

1 Introduction

LaTeX 2_ε 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.

Vous connaissez peut-être des astuces pour définir des commandes avec plusieurs arguments optionels, ou avec l'argument optionel en dernier. Mais utiliser ces astuces vous force à résoudre certains problèmes (ordre d'utilisation des arguments, gestions des espaces) qui peuvent s'avérer délicats. C'est n'est de toute façon pas le genre de choses auxquelles vous devriez avoir à réfléchir quand vous rédigez un document.

L'extension xargs vous fournit donc un moyen pratique et (je l'espère) robuste de définir de telles commandes, avec une syntaxe intuitive de la forme $\langle clé \rangle = \langle valeur \rangle$.

2 Usage

L'extension xargs fournit des analogues de toutes les macros de LaTeX 2_ε 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 : par exemple, `\newcommandx`, `\renewcommandx` ou encore `\newenvironmentx`. Elles ont par ailleurs toutes la même syntaxe. Je ne présenterai donc que `\newcommandx`.

Commençons par un exemple. Après la définition

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

vous pouvez utiliser la macro `\vect` d'une des façons suivantes :

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

Vous avez sans doute compris qu'il s'agit d'une macro prennant trois arguments au total, mais dont le premier et le troisième argument sont optionnels. Ils ont chacun une valeur par défaut (respectivement `1` et `n` ici). Vous pouvez également remarquer que la syntaxe de `\newcomandx` est très proche de celle de `\newcommand`, la seule différence étant qu'au lieu d'indiquer la valeur par défaut de l'unique argument optionel, vous indiquez ici le numéro de chaque argument à rendre optionel, suivi de sa valeur par défaut.

Détaillons maintenant la syntaxe de `\newcommandx`, qui, répétons-le, est aussi la syntaxe de toutes les autres commandes de l'extension xargs. (Pendant qu'on est dans les détails, voici la liste complète de ces commandes : `\newcommandx` et `\renewcommandx` pour les macros simples, `\providecommandx` pour s'assurer qu'une macro existe, `\DeclareRobustCommandx` pour (re)définir une macro robuste, `\CheckCommandx` pour vérifier le sens d'une macro, `\newenvironmentx` et `\renewenvironmentx` pour les environnements.)

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

Rappellons 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 entre crochets 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 est alors optionel. 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.

Concernant l'usage des commandes, notez que, si les arguments 1 et 2 (par exemple) sont optionnels, vous ne pouvez spécifier de valeur pour l'argument 2 si vous n'en avez pas spécifié pour l'argument 1. Ce comportement est cohérent avec celui de commandes LATEX existantes, comme `\makebox`. Il ne provient pas d'une limitation technique mais surtout de mon incapacité à imaginer une situation où il est vraiment gênant et une syntaxe intelligente pour le contourner. À ce sujet, observez comment j'ai pris soin de séparer les deux arguments optionels par l'argument obligatoire dans le définition de `\vect` ci-dessus.

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'acollades, il vous faudra écrire quelque chose comme `1={{{{\large blabla}}}}`.

La dernière particularité de la ⟨*liste*⟩ que vous devez connaître, est qu'elle ne doit pas contenir de `\par` (ou de ligne vide). C'est le seul point (à ma connaissance) sur lequel les macros d'xargs diffèrent des macros standard. Cette limitation est liée à un choix dans l'implémentation d'xkeyval, que j'ai eu un peu la flemme de contourner (en fait, je ne suis pas pleinement satisfait de la fiabilité des contournements que je connais). Vous pouvez utiliser `\endgraf` à la place de `\par` en cas de besoin. Si cette limitation vous gêne, merci de me le faire savoir.

Parlons maintenant de trois fonctionnalités d'xargs qui sont plus ou moins cachées mais que j'espère pratiques. La première est que les macros sont crées si possible de façon économique : si vous utilisez `\newcommandx` pour définir un macro que vous auriez pu définir avec `\newcommand`, xargs le détectera et utilisera automatiquement `\newcommand` à la place. Ainsi, vous n'avez à vous soucier de rien et vous pouvez utiliser tout le temps `\newcommandx` sans vous poser de questions.

La deuxième fonctionnalité est elle aussi transparente, mais comme j'ai pris la peine de la coder, il faut bien que j'en parle. Une macro crée par xargs n'avalera pas les espaces qui la suivent, même si le dernier argument est optionel et n'est pas spécifié. C'est un des avantages d'xargs sur les astuces classiques. Comparez les deux sur l'exemple (idiot) suivant :

```
\newcommand\compliment[1]{#1 est gentil\complinterne}
\newcommand\complinterne[1][le]{#1}
\newcommandx\complimentx[2][2=le]{#1 est gentil#2}
```

`\compliment{Clothilde} et\ldots`	Clothilde est gentilleet…
`\complimentx{Clothilde} et\ldots`	Clothilde est gentille et…

Observez comme l'espace a été avalé de façon indésirable dans le premier cas.

Enfin, les macros d'xargs essayent de se comporter en tous points comme leurs homoloques standard. Il y a deux exceptions : la première est que vous ne vous pouvez pas utiliser `\par` comme vous voulez, je l'ai dit plus haut, et je le regrette. La deuxième réside dans certains comportements de `\CheckCommandx`. En effet, à l'heure où j'écris ces lignes, `\CheckCommand` souffre de deux bugs (voir PR/3971) que j'espère avoir évités dans `\CheckCommandx`.

Vous savez maintenant 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. Si vous découvrez des bugs, merci de me les signaler. Si vous trouvez cette extension utile, ça me ferait aussi plaisir de le savoir.

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