LaTeX, créer ses commandes

Ou comment séparer encore un peu plus le fond de la forme

Bertrand Masson

3 juillet 2009

Introduction

Lorsque dans ton document un mot, un groupe de mots ou un objet (tableau, une boite, ...) reviennent plusieurs foi, il est toujours judicieux de créer une commande qui le reproduit et ce pour plusieurs raisons :

- tu gagnes du temps, il est plus facile d'écrire `\bsr` que bilan scientifique régional ;
- tu diminues les risques d'erreurs, les fautes de frappes : ton `\bsr` sera toujours écrit de la même façon ;
- si ton patron te dit eh coco c'est pas mal ton rapport mais il faudrait me mettre des majuscules à « Bilan Scientifique Régional » pas de problème 3 lettres à changer dans la définition de ta commande `\bsr` et les 45 « bilan scientifique régional » de ton texte se transforment en « Bilan Scientifique Régional »
- tu restes dans l'esprit LaTeX qui préconise la séparation du fond et de la forme. Tu restes concentré sur le fond et tu ne te poses plus la question à ta centième page de texte si au début tu as mis une majuscule à bilan ou pas ?

Pour cette fiche, on va rester sur des exemples simples de commande mais tu pourras avec l'expérience bâtir des commandes très sophistiquées avec des test conditionnels.

construire \bsr

Pour construire une commande tu utilises la commande `\newcommand` de la façon suivante : `\newcommand{nomDeTaCommande}{CodeLaTeX}`
Pour reprendre l'exemple de l'introduction :

```
\newcommand{\bsr}{bilan scientifique régional}
```

Voilà c'est pas plus compliqué que cela. Tu places cette définition dans le préambule de ton source LaTeX entre l'entête dans lequel tu charges les packages et le `\begin{document}`. Tu l'utilises comme toutes les commandes LaTeX en plaçant dans ton source `\bsr`. Après compilation tous les `\bsr` seront remplacés par bilan scientifique régional.

Une petite remarque sur la façon qu'a LaTeX de reconnaitre une commande

Pour LaTeX une commande commence toujours par un \ (c'est pour cette raison que c'est un caractère réservé et que tu ne peux le mettre dans ton document sauf en utilisant la commande `\textbackslash`). Ne l'oublie pas quand tu définis ta commande, tu dois écrire `\newcommand{\bsr}{bilan scientifique régional}` et non pas `\newcommand{bsr}{bilan scientifique régional}`
Pour la fin de la commande c'est un peu plus compliqué. Pour LaTeX la fin d'une commande est indiqué soit par une espace soit par {}. Si tu utilise l'espace, celle-ci est « mangée » par LaTeX car considérée comme une fin de commande et pas comme une espace entre 2 mots. Bon je me rend compte que ce que je raconte n'est pas très clair, mais des exemples vont te faire comprendre cette subtilité LaTeXiène. Prenons le cas du oe collé comme dans cœur. Pour LaTeX la commande pour écrire un oe collé est `\oe`. Si tu écris `c\oeur` LaTeX va te retourner une erreur car il ne connait pas la commande `\oeur`. Tu dois écrire `c\oe ur` ou `c\oe{}ur`. Moi je préfère la deuxième solution, que je trouve plus claire.

Une petite remarque, suite

Examinons maintenant le cas du mot LaTeX qui s'écrit à l'aide de la commande `\LaTeX` (Attention aux majuscules). Voici différentes façon de l'écrire en fonction de sa position dans la phrase :

- `\LaTeXest fantastique.` provoque une erreur de compilation car la commande `\LaTeXest` n'est pas reconnue ;
- `\LaTeX est fantastique.` = LaTeXest fantastique. Pas d'erreur de compilation mais LaTeX collé au est car l'espace considérée comme une fin de commande est mangée (l'espace séparant 2 mots est un caractère typographique féminin : une espace) ;
- `\LaTeX{} est fantastique.` = LaTeX est fantastique. Ce que l'on recherche ;
- `\LaTeX  est fantastique.` = LaTeXest fantastique. Là j'ai mis 2 espaces après `\LaTeX`. Je me suis dit la première va servir de fin de commande et l'autre de séparation, et bien ça ne marche pas car LaTeX ne reconnait pas la multiplication des espaces. Une, 2 ou 10 espaces pour lui c'est pareil il n'en reconnait qu'une seule, donc ici on se retrouve dans le même cas de figure que le deuxième exemple.

Une petite remarque, ça continue

On ne retrouve pas le même problème lorsque le mot LaTeX termine une phrase, ou se trouve juste avant une virgule, ou tous autres signes de ponctuation. En effet un nom de commande ne peux contenir que des lettres, donc quand LaTeX rencontre un point un chiffre, un tiret, ... il considère que l'on n'est plus dans un nom de commande, donc les expressions suivantes donne le même résultat :
`\LaTeX .` = LaTeX.
`\LaTeX.` = LaTeX.
`\LaTeX{}.` = LaTeX.
La commande `\LaTeX56` ne retourne pas d'erreur de compilation et donne : LaTeX56 et
`\LaTeX!` (sans espace entre le X et le !) donne LaTeX ! (avec espace entre le X et le !)
Tu suis toujours ? En tout cas, moi pour pas me prendre la tête, je saisis toujours mes commandes en les terminant par {}, comme ça je suis sur de ne pas me planter. Mais tu fais comme tu veux.

Une petite remarque (finalement j'aurais du mettre du pluriel), mais là c'est fini

Un dernier example le cas des guillemets français qui s'écrivent sous LaTeX avec l'option frenchb du package babel (`\usepackage[frenchb]{babel}`, c'est pour écrire en français) `\og` (pour **o**uvrez les **g**uillemets) et `\fg` (pour **f**ermez les **g**uillemets). Tu peux écrire `\og un mot` = « un mot, tu obtiens ce que tu veux car l'espace après le `\og` indique bien une fin de commande et est mangée, mais la commande `\og` retourne un « et une espace, on a donc bien notre espace pour séparer le guillemet ouvrant de notre mot. Sympa les programmeurs. Mais attention ça ne marche pas pour le `\fg` qui se comporte comme la commande `\LaTeX` vue ci-dessus, pour la simple et bonne raison que `\fg` peut être suivit d'une espace ou d'une ponctuation. La commande `\fg` ne retourne que » sans espace. Tu dois donc écrire par exemple pour des guillemets en fin de phrase : `\fg .` ou `\fg{}.` ou `\fg.` et `\fg{} un mot` quand `\fg{}` précéde un mot. Pas cons ces programmeurs.

Ah au fait pour écrire LaTeXiène, j'ai tapé cette commande `\LaTeX ième`.

Le package xspace

Pour résoudre ces problèmes d'espace, il existe le package xspace (`\usepackage{xspace}`). Attention il concerne les commandes que tu vas créer, il est sans effet sur celle interne à LaTeX, sauf pour `\fg` qu'il rend intelligent et capable de savoir s'il est suivit par une espace et un mot ou un si il précède une ponctuation. Dans ce cas tu peux écrire `\fg un mot`, donne » un mot. Bon j'arrête la mes digressions et on revient à la création de nos commandes. On va créer la commande `\n` qui écrit le mot noir.

```
\newcommand{\n}{noir}
```

Selon les remarques précédentes du dois écrire dans ton source `Le chat est \n.` ou `Le chat est \n{}.` et `Il est \n{} et poilu.` pour obtenir les résultats escomptés.

Le chat est noir.
Il est noir et poilu.

Maintenant si tu as chargé le package xspace tu peux utiliser la commande xspace de cette façon : `\newcommand{\n}{noir\xspace}`. Le problème de l'espace est réglé, tu peux écrire `Le chat est \n.` et `Il est \n et poilu.` et tu obtiens le même résultat :

Le chat est noir.
Il est noir et poilu.

Les commandes à arguments

Voila comment elle se présente, c'est la même commande que précédament à laquelle on ajoute une option (entre [ ] comme toutes les options en LaTeX) : `\newcommand{nomDeTaCommande}[nombreArguments]{CodeLaTeX}`. On passe tout de suite à un exemple. On va créer une commande pour écrire les surfaces.

Encore un aparté

C'est fou ce qu'il y a comme aparté dans cette fiche. Je préfère expliquer toutes les commandes que j'utilise même celles qui ne sont pas liées directement au sujet de la fiche, sinon tu vas te demander c'est quoi cette commande `\numprint` qu'il utilise pour la surface et qu'est-ce qu'elle vient faire là ? Tu pourrais m'objecter que j'étais pas obligé de l'employer, l'exemple aurait été compréhensible sans. Certes mais quitte a créer une commande autant le faire du mieux possible. En typographie française on utilise une espace fine insécable pour séparer les milliers. On n'écrit pas 1256 ou 125987 mais 1 256 et 125 987, c'est le rôle de la commande `\numprint` du package du même nom (`\usepackage{numprint}`, `\numprint{125987}`). Je vais également utiliser la commande `\up` qui met en exposant (pas de package à charger).

Notre commande surface

Bon revenons à notre commande `\surf`, voici la méthode pour la créer :
`\newcommand{\surf}[1]{\numprint{#1}\,m\up{2}}`
Voici comment tu l'utilises et le résultat : `\surf{1250}` = 1 250 m^2
Un argument c'est l'équivalent d'une variable dans un langage de programmation. Le [1] indique le nombre d'arguments que contient la commande, ici il n'y en à qu'un. Tu ne peux utiliser que 9 arguments dans une commande, c'est une limitation qui peut-être gênante, voila, LaTeX n'est pas parfait :-(#1 est le nom de la variable, si tu en utilises plusieurs elles seront toutes écrites de cette façon #2,#3,. . .,#9. Le `\,` crée une espace fine insécable pour éviter que le nombre et le m^2 se retrouve séparés sur deux lignes (et oui en typographie française les nombres et les unités sont séparés par une espace fine insécable et non une espace insécable obtenue par ~).
Une petite remarque en passant. Dans les commandes à arguments tu n'as plus à te soucier des fins de commandes et `\xspace` ne sert à rien car la commande à arguments se termine de toute façon par une `}`

Une commande à plusieurs argument

On va fabriquer une commande bidon qui permet de créer une phrase. C'est complètement idiot et sans intérêt comme commande, mais ça va nous permettre d'étudier plusieurs comportements des commandes. Notre commande va s'appeler `\phrase` et va comporter 3 arguments, l'un pour le sujet, l'autre pour le verbe et le troisième pour le complément. Voila sa construction : `\newcommand{\phrase}[3]{#1 #2 #3.}` et son utilisation : `\phrase{Le chat}{est}{\n}` qui donne :
Le chat est noir.
Tu n'as pas oublié la commande `\n` crée précédemment ? Maintenant observe le comportement de `\newcommand{\phraseb}[3]{#1 #3 #2.}` :
`\phraseb{Le chat}{est}{\n}` = Le chat noir est.
Donc attention à l'ordre des arguments.
On peut décider que le sujet sera toujours au féminin singulier et le complément un lieu précédé de « sur ».
`\newcommand{\phrasec}[3]{La #1 #2 sur #3.}` qui donne :
`\phrasec{maison}{posée}{la colline}` = La maison posée sur la colline.

Une commande avec un argument à valeur par défaut

On reste avec notre commande `\phrase` mais on désire dans notre phrase évoquer les chats, pas toujours, mais souvent. On va donc créer un argument ayant chat comme valeur par défaut.

```
\newcommand{\phrased}[3][Le chat]{#1 #2 #3.}
```

`\phrased{est}{\n}` = Le chat est noir.

Les chats c'est sympa mais j'aimerais évoquer le cas d'un chien bizarre. Pas de problème, il suffit d'utiliser la commande précédente comme ça :

`\phrased[Le chien]{est}{jaune}` = Le chien est jaune.

Attention aux [ ] le premier argument étant devenu facultatif on utilise des [ ]

On va encore tomber sur une limite de LaTeX. Tu ne peux avoir qu'un seul argument prédéfini par commande et c'est toujours le #1. La première restriction est ennuyeuse, mais il faut faire avec, par contre on peut détourner la seconde. Finalement c'est le verbe être qui est plus fréquemment l'invariant, donc on va modifier notre commande comme suit :

`\newcommand{\phrasee}[3][est]{#2 #1 #3.}`. Tu vois c'est simple il suffit de modifier la position du #1. Le résultat :

`\phrasee{Le chat}{\n}` = Le chat est noir.

Pour en finir avec les commandes avec un argument

Tu peux répéter un même argument plusieurs fois dans ta commande :
`\newcommand{\phrasef}[3]{#1 #3 #2 #3, #3 c'est #3, il n'y a\dots}`
`\phrasef{Le chat}{est}{\n}` qui donne :
Le chat noir est noir, noir c'est noir, il n'y a...
Bien entendu tu n'es pas obligé de remplir tous les « champs » d'une commande, mais dans ce cas il faut quand même qu'il y ai le bon nombre de {} sinon tu te retrouves avec une erreur de compilation. Quelques exemples :
`\phrase{Le chat}{}{\n}` = Le chat noir.
La commande de la page précédente :
`\phrasee[]{Le chat}{\n}` = Le chat noir.
Eh merd... ça marche pas ! On se retrouve avec 2 espaces entre chat et noir. En effet dans ce cas on a un chat, une espace, un rien, une espace, noir. En LaTeX rien c'est quand même quelque chose et donc nos 2 espaces ne sont pas contiguë et donc LaTeX ne les fusionne pas et tu te retrouves avec 2 espaces.
La solution ? utiliser `\ignorespaces`
`\newcommand{\phraseg}[3]{#1 \ignorespaces #2 #3.}`
`\phraseg{Le chat}{}{\n}` = Le chat noir.

Le package xargs

Cette limitation du nombre d'arguments optionnels est pénible. Réjouis-toi, Manuel Pégourié-Gonnard a concocté un package qui permet de s'affranchir de cette limite : le package xargs. Son utilisation est très simple et ne te dépayseras pas :

```
\newcommandx{\nomCommande}[nombreArguments][liste clé=valeur]{codeLaTeX}
```

(n'oublies pas le x à \newcommandx). Si on reprend notre commande phrase, avec 2 arguments optionnels et 1 obligatoire :

`\newcommandx{\phrasex}[3][1=Le chat,3=noir]{#1 #2 #3.}`, en voici plusieurs utilisation :

`\phrasex{est}` = Le chat était noir.

`\phrasex[Le chien]{est}` = Le chien est noir.

`\phrasex{est}[jaune]` = Le chat est jaune.

`\phrasex[Le chien]{est}[jaune]` = Le chien est jaune.

Attention à la place de chaque éléments. Si tu crées une commande avec 1 seul argument optionnel : `\newcommandx{\phrasexb}[3][3=noir]{#1 #2 #3.}`

`\phrasexb[jaune]{Le chien}{est}` = [j noir.aune]Le chienest

Une commande grandeur nature

Maintenant on va mettre en application tout ce que l'on a appris pour créer une vraie commande, avec de vrais morceaux de code dedans et qui est utile. Enfin à moi elle est utile. On va écrire une commande pour gérer l'écriture des siècles, un truc du genre `\siecle{17}` qui donne XVII^{e} siècle.
Pour ce faire on va avoir besoin de la commande `\up` que l'on à déjà vue, de la commande `\romannumeral` qui transforme un nombre en chiffres arabe en nombres en chiffres romains `{\romannumeral 128}` = cxxviii. Comme tu peux le voir elle écrit en minuscule, donc il va nous falloir aussi `\uppercase` qui met en majuscule `\uppercase{cxxviii}` = CXXVIII et on a besoin également de `\expandafter`, c'est une commande qui... Je crois que je suis obligé de faire un nouvel aparté.

Encore un aparté

A vrais dire je n'ai pas compris à quoi sert la commande `\expandafter`, mais `\uppercase{\romannumeral 128}` ne marche pas il faut utiliser `\uppercase\expandafter{\romannumeral 128}`. `\expandafter` est une commande TeX. N'ayant jamais lu le TeXbook je ne sais pas ce qu'elle fait et une recherche sur internet ne m'a pas vraiment aidé :« sert à placer des tokens dans le flot d'entrée de TeX, cela sert à calculer une séquence de tokens avant de la faire interpréter. ». Donc pour l'instant j'applique bêtement. Surtout que dans la commande que j'utilise quotidiennement je ne me sers pas de `\uppercase` mais de `\textsc` qui mets en petites capitales qui je trouve sont plus jolie pour indiquer les siècles. Seulement voilà, la police par défaut de beamer n'a pas de petites capitales et LaTeX les remplace par une police roman comme ça : XVII^{e} siècle. C'est pas terrible ! Je pourrait changer la police de beamer avec `\usepackage{helvet}`, mais je n'aime pas helvetica pour la projection ou la lecture sur écran, elle est trop grasse, trop noire, je la trouve agressive. J'aime bien la police par défaut de beamer, voila pourquoi on se retrouve avec `\expandafter`.

Une commande grandeur nature

Revenons à nos moutons. Voici la version « beamer » de la commande `\siecle`. C'est une commande avec 1 seul argument (le chiffre que l'on souhaite transformer). Après les explications ci-dessus tu ne dois pas avoir de difficulté à la comprendre. Le `~` insère une espace normale insécable.

```
\newcommand{\siecle}[1]{\uppercase\expandafter{\romannumeral #1}\up{e}~siècle}
```

Et voici la version petites capitales

```
\newcommand{\siecle}[1]{\textsc{\romannumeral #1}\up{e}~siècle}.
```

Et voilà tu peux y aller à fond :

`\siecle{23}` = XXIII^{e} siècle

`\siecle{624}` = DCXXIV^{e} siècle

`\siecle{2596}` = MMDXCVI^{e} siècle

`\siecle{1}` = I^{e} siècle

Oups ! ! ! ça coince 1^{e} en typographie française ça s'écrit 1er. Pas grave on va faire une nouvelle commande pour gérer le cas du 1.

Je ne pense pas que se soit une bonne idée, entre taper `\siecleUn{1}` et `I\up{er} siècle` le gain de performance ne saute pas au yeux, en plus il ne faut pas oublier de changer de commande à chaque fois. Il est préférable d'améliorer notre commande `\siecle`. Pour cela il va falloir tester si la variable `#1` vaut 1 ou non.

Une boucle conditionnelle

On va utiliser la boucle suivante :
`\ifnum #1=1` début de la boucle qui teste si notre variable est égale à 1.
codeLaTeX je mets ici les commandes si `#1` vaut 1
`\else` sinon
codeLaTeX je mets ici les commandes si `#1` est différent de 1
`\fi` fin de la boucle
Voici notre commande :

```
\newcommand{\siecle}[1]{%
\ifnum #1=1%
\uppercase\expandafter{\romannumeral #1}\textsuperscript{er}~siècle%
\else%
\uppercase\expandafter{\romannumeral #1}\up{e}~siècle%
\fi}
```

Maintenant `\siecle{1}` = I^{er} siècle
– C'est quoi ces % que tu as mis partout, tu n'en a pas parlé des % ?
C'est parti pour un nouvel aparté.

Encore un aparté : les %

J'aurais pu écrire la commande sur une seule ligne,sans %, mais la lisibilité n'aurais pas été terrible. La commande est écrite sur plusieurs lignes uniquement pour faciliter la lecture. En LaTeX un saut de ligne insère une espace. Je rappelle que pour sauter une ligne il faut 2 sauts de ligne ou utiliser la commande `\par`. Les % indiquent à LaTeX d'ignorer ce qui suit, ce qui est pratique pour insérer des commentaires. Ici ils obligent LaTeXà ignorer les sauts de ligne et donc à ne pas ajouter d'espace. Pour cette commande l'ajout d'espace n'est peut-être pas génant mais pour certaine ça peut l'être. Allez un petit exemple. Je vais reprendre notre commande `\bsr` du début et l'écrire sur plusieurs lignes

```
\newcommand{\bsrb}{
bilan
scientifique régional
}
```

La commande (`\bsr`) donne :
(bilan scientifique régional) : avec la commande écrite sur une seule ligne
(bilan scientifique régional) : avec la commande écrite sur plusieurs lignes

Des exemples

Voici une petites série d'exemple, ils sont extraits de mes documents professionnels donc ne te préoccupe pas du jargon archéologique. Tout d'abord j'emploie énormément d'abréviation, comme `\bsr` qui me facilite la saisie, puis aussi beaucoup d'autre pour garantir l'homogénéité de mon document, par exemple il y a plusieurs façon d'écrire « avant Jésus Christ », une commande résout le problème.

```
\newcommand{\apjc}{apr. J.-C.\xspace}
\newcommand{\avjc}{av. J.-C.\xspace}
\newcommand{\carbon}{\up{14}C\xspace}
\siecle{3}\avjc
```

= IIIe siècle av. J.-C.

LaTeX possède beaucoup de commandes internes, il peut arriver que le nom que tu choisisses soit attribué dans ce cas LaTeX te prévient lors de la compilation par une remarque explicite.

Des exemples, suites

Il m'arrive souvent d'avoir besoin du même tableau dans lequel seules les données changent. Je ne vais pas d'écrire la commande mais juste te montrer comment je l'utilise.

```
\newcommand{\tabIndice}[4]{%
\begin{tabular}{|l|c|}
\hline
{\sffamily \color{gray!80}IL}&$#1$\\
\hline
{\sffamily \color{gray!80}IF}&$#2$\\
\hline
{\sffamily \color{gray!80}IFs}&$#3$\\
\hline
{\sffamily \color{gray!80}ILam}&$#4$\\
\hline
\end{tabular}
}
```

`\tabIndice{23,5}{12,2}{4,2}{11,5}` donne

IL	23,5
IF	12,2
IFs	4,2
ILam	11,5

Comment LaTeX interprète une commande

LaTeX ne résout pas d'abord la commande puis insère le résultat dans ton document. Il ne fait que remplacer `\taCommande` par les lignes de codes qu'elle contient et ensuite le document est compilé. Donc le même groupe d'instructions peut être répartis dans plusieurs commandes. Je me rend compte que ce n'est pas très clair et qu'un exemple vaudra mieux.

Exemple

On va reprendre le cas du tableau, mais cette fois ci, on connait le nombre de colonne mais pas le nombre de lignes qui va varier en fonction des données disponibles. On va donc créer 2 commandes. La première pour l'entête du tableau qui est invariant la seconde pour les lignes (donc les données). Voici la commande pour l'entête du tableau :

```
\newcommand{\entete}{%
\begin{tabular}{%
|m{0.2\linewidth}|>{\centering}m{0.07\linewidth}|m{0.2\linewidth}|}
\hline%
\rowcolor{gray}%
\color{white} \bfseries Matière%
&%
\color{white} \bfseries Réf.%
&%
\color{white} \bfseries Couleur\\%
\hline%
}%
```

La suite du tableau

On va maintenant créer une commande pour la saisie des données :

```
\newcommand{\donnees}[3]{%
\textbf{#1}&#2&#3\\
\hline%
}
```

Voila il ne te reste plus qu'as utiliser les commande comme ça :

```
\entete
\donnees{Silex}{Z2}{Bleu gris}
\donnees{Silex}{Z3}{Noir}
```

Et tu te retrouve avec un message d'erreur ! Et oui un tableau se termine par `\end{tabular}`. Il faut donc écrire :

```
\entete
\donnees{Silex}{Z2}{Bleu gris}
\donnees{Silex}{Z3}{Noir}
\end{tabular}
```

Le résultat

```
\entete
\donnees{Silex}{Z2}{Bleu gris}
\donnees{Silex}{Z3}{Noir}
\end{tabular}
```

Matière	Réf.	Couleur
Silex	Z2	Bleu gris
Silex	Z3	Noir

```
\entete
\donnees{Silex}{Z2}{Bleu gris}
\donnees{Silex}{Z3}{Noir}
\donnees{Grès}{B23}{Beige}
\end{tabular}
```

Matière	Réf.	Couleur
Silex	Z2	Bleu gris
Silex	Z3	Noir
Grès	B23	Beige

Conclusion

Voila un petit aperçu de la création de commande. Le sujet est vaste, je n'ai pas parlé de la création d'environnement, de la possibilité de redéfinir des commandes LaTeX avec `\renewcommand` du package `ifthen` qui apporte de nombreuses structures conditionnelles,... De nouvelles fiches en perspectives.
Bébert