LaTeX, un peu, beaucoup

Pascal Dupont

Mots clés : LaTeX, macro

5. Les macros

Les macros (ou plus exactement *macro-commandes*), pour le dire simplement, ce sont des abréviations réutilisables à volonté.

Il est possible d'en introduire dans tout document, à n'importe quel moment, mais pour les plus utilisées d'entre elles, il vaut mieux créer un fichier où les stocker une fois pour toutes, ou à tout le moins jusqu'à ce que le besoin de les modifier se fasse sentir.

Signalons aussi d'emblée que, comme les commandes prédéfinies, les macros que nous construirons peuvent ou non avoir des arguments.

Pourquoi utiliser des macros ?

Il y a différentes justifications pour introduire des macros dans son document, meilleures les unes que les autres. Il y a aussi de mauvaises raisons, c'est-à-dire des choses à ne pas faire. Commençons par celles-ci.

Les créateurs de TEX et de LATEX, Donald Knuth et Leslie Lamport, ont fait le choix de donner aux commandes prédéfinies dans le langage des noms explicites, même s'ils doivent être un peu longs ; par exemple `\Leftrightarrow` pour obtenir une double flèche « si et seulement si » ou `\displaystyle` pour mettre une équation en style hors-texte. L'utilisateur pourrait regretter ce choix et décider d'introduire des abréviations pour ces noms de commande un peu longuets (et en anglais de surcroit). Cependant, ce serait une erreur, à plusieurs égards : un document ainsi codé serait moins lisible et plus difficile à transmettre à un tiers ; en outre, la syntaxe globale du langage perdrait en cohérence : comment retenir *quelles* commandes bénéficient d'une abréviation ? N'introduisons donc pas de synonymes par simple paresse dactylographique.

En revanche, voici des raisons constructives d'introduire des macros.

1. Lorsqu'il s'agit d'une construction assez élaborée, qui fait p. ex. déjà appel elle-même à d'autres commandes, et qui servira plus ou moins souvent. Par exemple, j'ai pour marotte de distinguer (si elles existent, naturellement) l'inverse f^{-1} et la réciproque $\overset{-1}{f}$ d'une fonction f. Obtenir de LATEX qu'il place le « -1 » au-dessus du nom de la fonction et non en exposant, à droite, si ce n'est pas du grand art, nécessite tout de même un peu de travail. Autant donc que ce travail soit effectué une fois pour toutes : utilisons une macro `\rcpq`, laquelle admettra pour argument le nom de la fonction à réciproquer.

2. Lorsqu'il s'agit de se choisir une notation qui sera figée à travers tout un texte voire un ensemble de textes, mais qui est susceptible d'être modifiée à terme.
Un exemple typique est celui des ensembles de nombres, par exemple celui des réels. À la suite d'illustres prédécesseurs, je prône de le noter $\mathbf{R}$ (comme en gastronomie, bannissons le « faux gras » $\mathbb{R}$, qui n'est qu'un pâle substitut, à n'utiliser que lorsque les caractères gras ne sont pas disponibles, en particulier dans les manuscrits ou les textes dactylographiés).
Il va de soi que pour éviter de taper un grand nombre de fois `\mathbf R`, j'ai défini une macro `\RR`. Mais ce faisant, ne suis-je pas en contradiction avec la règle d'éviter les macros « de pure paresse » ? Non ; il y a ici une véritable valeur ajoutée, qui est celle-ci : le jour où un de mes livres, p. ex., est publié dans une collection dont le directeur m'*impose* une convention autre que la mienne, il n'est pas besoin de remplacer tous les `\mathbf R` du texte par des `\mathbb R` (même avec le « Remplacer tout » de l'éditeur de texte, ce n'est pas toujours si évident !) ; il suffit, au lieu de cela, de modifier, à un seul endroit (dans le préambule du document ou dans le fichier de commandes), la définition de la macro.

3. Lors de la rédaction d'un gros ouvrage, qui s'étale peut-être sur plusieurs années, et qui est susceptible de pauses plus ou moins longues, le risque

existe de ne plus se souvenir de la manière dont on a noté telle ou telle notion. Par exemple, si je ne risque guère d'oublier que mon habitude est de noter $\mathcal{A}(P)$ l'aire du polygone P, comment diable ai-je désigné, il y a six mois, son aire orientée ? $\mathcal{A}_o(P)$? $\mathcal{A}_{\text{or}}(P)$? $\mathcal{A}^{\pm}(P)$? À l'intérieur d'un syllabus, il est de bon ton de rester cohérent… Évidemment, le retrouver ne devrait pas être trop difficile ; mais si j'ai pris le soin de définir une macro `\aireorientee` — avec, comme de juste, un nom bien explicite —, je ne risque pas de l'oublier.

Comment et pourquoi se constituer un fichier de commandes ?

Une fois que vous vous êtes constitué un petit capital de macros, accumulées dans le préambule de l'un de vos documents, il suffit évidemment de simples « Copier-Coller » pour l'avoir à disposition dans toute votre production ultérieure. Une saine gestion des ressources demanderait que vous ne recopiiez que celles des macros qui servent effectivement dans le nouveau document, mais le gaspillage qui consiste à les reprendre toutes par principe est insignifiant en temps de calcul (la compilation, c'est un calcul).

Il existe cependant une meilleure habitude à prendre : se créer une fois pour toutes (mais en restant prêt bien sûr à toute amélioration) un document de base, consistant essentiellement en un préambule, qui sera cloné puis complété en fonction des besoins chaque fois qu'un nouveau document TEX doit être créé.

La solution optimale, cependant, est de ranger toutes nos définitions de macros dans un document, que nous nommerons, par exemple, *MacrosPersonnelles.tex*, et qui sera appelé par un `\input{MacrosPersonnelles.tex}` dans le préambule de tout nouveau document TEX.

Attirons cependant l'attention sur une petite précaution à prendre. Notre fichier de macros est appelé à évoluer, parce que nos besoins vont sans doute s'accroitre, mais aussi peut-être parce que nos préférences de notations vont changer, ou parce que nous mettrons au point de meilleures solutions pour résoudre nos problèmes.

Or, si nous nous contentons de modifier notre document *MacrosPersonnelles.tex*, il nous sera impossible de recompiler à l'identique nos documents anciens : les nouvelles définitions de macros auront remplacé les anciennes. Le plus sage est donc sans doute de dater notre ficher de macros en le nommant *MacrosPersonnelles20140905.tex* s'il a été créé le 5 septembre 2014 ([1]), et en adaptant à la date de modification les versions modifiées ultérieures (sans supprimer les versions précédentes) ; en l'appelant par un `\input{MacrosPersonnelles20140905.tex}`, nous sommes sûrs que toute compilation future produira le même résultat que la compilation originale : si entretemps des macros ont été modifiées, c'est dans un nouveau fichier. Évidemment, si notre souhait, à un certain moment, est de prendre en compte la nouvelle version des macros, c'est possible : il suffit de modifier en conséquence le nom du document appelé.

Comment définir des macros ?

Après ces longs préliminaires, nous n'avons encore rien dit du cœur de notre sujet : à quoi ressemble la définition d'une macro ?

Cette question admet deux réponses, selon que l'on utilise la syntaxe de TEX « pur » ou la syntaxe de LATEX. Rappelons en effet que LATEX est une « surcouche » qui s'ajoute à TEX, avec pour conséquence que les commandes TEX sont toujours disponibles lorsque nous travaillons en LATEX.

Dans cette liste d'articles, c'est principalement LATEX que nous décrivons, et donc nous recommandons de travailler « à la LATEX ». Toutefois, à la fin de cette section, nous dirons quelques mots de la manière de définir des macros « à la TEX ».

Commençons par le cas le plus simple, celui des macros sans argument. Par exemple, que faut-il faire pour que `\RR` dans le document source provoque l'apparition de **R** dans le document compilé et non un message d'erreur dans le journal de la compilation, expliquant qu'à telle ligne le compilateur a rencontré une commande inconnue ?

Il suffit de placer, n'importe où avant la première utilisation de la macro (mais le plus propre est évidemment de la placer dans le préambule), l'instruction

```
\newcommand{\RR}{\mathbf{R}} ;
```

c'est donc la commande `\newcommand` qui définit de nouvelles commandes ; elle admet deux arguments :

([1]) Ce n'est pas pour nous aligner servilement sur les Anglosaxons que nous suffixons le nom du fichier de la date écrite sous la forme année-mois-jour ; c'est pour que le classement alphabétique des documents corresponde au classement chronologique.

d'abord le nom de la nouvelle commande, qui devra obligatoirement commencer par une controblique, et ensuite, le « développement » de la commande, c'est-à-dire ce par quoi le compilateur remplacera le nom de la macro lors de la compilation. Ainsi, à la compilation de

```
\documentstyle{article}
\newcommand{\RR}{\mathbf{R}}
\begin{document}
$\RR$ est l'ensemble des réels.
\end{document},
```

tout se passera comme si notre document source contenait

```
\documentstyle{article}
\begin{document}
$\mathbf{R}$ est l'ensemble des réels.
\end{document}.
```

Ceci appelle néanmoins quelques remarques.

1. Rappelons d'abord que le choix de minuscules ou de capitales dans les noms de commandes a de l'importance ; si nous avons défini la commande `\RR`, la commande `\rr` n'existe pas pour autant, et si nous tapons, par distraction, `\rr` dans notre document source, nous n'obtiendrons rien d'autre qu'un message d'erreur.

2. Nous avons écrit plus haut que la définition de la commande devait se trouver n'importe où avant sa première utilisation ; ce n'est pas tout à fait exact. En effet, les définitions des commandes sont *locales*, ce qui signifie que si une telle définition se trouve dans un groupe (c'est-à-dire entre deux accolades appariées, ou entre le `\begin{...}` et le `\end{...}` délimitant un environnement), elle perd tout effet une fois l'environnement refermé. Donc

```
\documentstyle{article}
\begin{document}
\newcommand{\RR}{\mathbf{R}}
\begin{center}
\textbf{Les ensembles de nombres}
\end{center}
$\RR$ est l'ensemble des réels.
\end{document}
```

donnera le résultat souhaité, tandis que

```
\documentstyle{article}
\begin{document}
\begin{center}
\newcommand{\RR}{\mathbf{R}}
\textbf{Les ensembles de nombres}
\end{center}
$\RR$ est l'ensemble des réels.
\end{document}
```

provoquera une erreur de compilation : la définition de la macro se trouve dans l'environnement *center* utilisé pour le titre, et donc est inconnue en dehors de cet environnement, en particulier lors de son utilisation à la ligne 7.

Mais répétons encore que, sauf raison particulière, la place « normale » des définitions de macros est dans le préambule.

3. Lorsque nous utilisons `\newcommand` pour définir une nouvelle commande, LaTeX, qui n'est pas tombé de la dernière pluie, vérifie que le nom que nous attribuons à notre macro n'est pas déjà utilisé ; des effets catastrophiques (et bien difficiles à diagnostiquer) pourraient en effet se produire si, par inadvertance, nous réutilisions le nom d'une commande qui existe déjà. Qui peut se vanter de connaitre *toutes* les commandes de LaTeX ? Il y en a par exemple qui sont cachées à l'intérieur d'un module et uniquement utilisées à l'intérieur de celui-ci !

Donc, si nous utilisons `\newcommand` en lui donnant comme premier argument le nom d'une commande existante, le compilateur s'arrêtera et produira un message d'erreur signalant le problème. Si c'est par étourderie que nous avons choisi ce nom pour notre macro, il suffit d'en imaginer un autre.

Mais il peut aussi arriver que, en connaissance de cause, nous souhaitions *re*définir une macro, c'est-à-dire en modifier la définition. C'est en effet possible : dans ce cas, au lieu de `\newcommand`, c'est `\renewcommand` qu'il faudra utiliser, selon les mêmes modalités — et celle-ci nous enverra sur les roses si la commande n'existe pas, comme de juste !

Les macros sans argument ne répondent cependant pas à tous les besoins. Par exemple, la macro `\rcpq` que nous voulons construire devra recevoir comme argument le nom de la fonction à réciproquer : le document source devra contenir `$\rcpq{f}$` pour produire $\overset{-1}{f}$.

La syntaxe pour obtenir une macro avec arguments (jusque 9) est

```
\newcommand{Nom}[NbArg]{Dévlpmt}
```

où : *NbArg* est le nombre d'arguments ; *Nom* est le nom de la macro (avec sa controblique initiale) ; *Dévlpmt* est le développement de la macro, dans lequel les 1er, 2e, ... arguments sont notés `#1`, `#2`, ... La même chose existe bien sûr avec `\renewcommand`.

Par exemple, une macro destinée à produire la valeur absolue d'un réel (ou le module d'un complexe) pourrait être définie comme suit :

`\newcommand[1]{\abs}{\left|#1\right|}`.

Après cette définition, nous pouvons taper

`$\abs{-2}=\abs{-\frac{3}{2}}+\frac{1}{2}$`

pour obtenir : $|-2| = \left|-\frac{3}{2}\right| + \frac{1}{2}$ (noter que grâce à l'usage des commandes `\left` et `\right`, la taille des délimiteurs (les barres verticales) s'adapte à leur contenu).

Mieux encore, il est possible de définir une nouvelle commande qui admet un argument optionnel. Par exemple, TEX possède l'opérateur `\lim`, qui produit « lim » ; le texte souscrit du type $x \to 0$ doit lui être ajouté comme un indice. Mais nous pourrions avoir envie de simplifier cela, en particulier si nous comptons encoder de longues listes d'exercices de calculs de limites. Notre but est de définir un opérateur `\Lim`, qui admettra deux arguments : le nom de la variable et le point vers lequel elle tend ; mais, comme la variable est « très souvent » x, nous souhaiterions nous faciliter la vie en déclarant ce premier argument optionnel, avec x comme valeur par défaut ; ainsi, nous ne devrons le fournir que lorsque la variable est autre que x. La définition suivante remplit ce contrat :

```
\newcommand{\Lim}[2][x]
  {\lim_{#1\rightarrow#2}}.
```

Pour obtenir $\lim_{t\to 0} f(t)$, nous tapons alors `$\Lim[t]{0} f(t)$` (ou même, plus concisément encore, `$\Lim[t]0f(t)$`), tandis que pour obtenir $\lim_{x\to 0} f(x)$, il nous suffit de taper `$\Lim{0} f(x)$` (voire `$\Lim0f(x)$`).

De manière générale, la syntaxe est

`\newcommand{`*`Nom`*`}[`*`NbArg`*`][`*`Dft`*`]{`*`Dévlpmt`*`}`

où, en plus ce ce qui a déjà été expliqué plus haut, *`Dft`* est la valeur par défaut du premier argument ; et de même pour `\renewcommand`. Bien remarquer qu'un seul argument optionnel est permis, le premier, qui sera donné, s'il échet, entre crochets, et que *`NbArg`* est le nombre *total* d'arguments, argument optionnel compris.

En TEX de base, les macros se définissent au moyen de la commande `\def` ; à titre d'exemple, l'équivalent des définitions de `\RR` (sans argument) et de `\abs` (macro à un argument) que nous avons données plus haut serait

```
\def\RR{\mathbf{R}}
```

et

```
\def\abs#1{\left|#1\right|}.
```

Cette commande `\def` ne vérifie pas si la commande existe déjà, ce qui prive son utilisateur d'un garde-fou par rapport à celui de `\(re)newcommand` ; elle ne permet pas, en outre, de définir facilement des macros avec argument optionnel. Cependant, elle admet des variantes puissantes et bien utiles, que nous ne décrirons pas ici.

Quelques macros utiles

À titre d'exemple, disons quelques mots du problème des intervalles, qui provient de ce que, si les Anglosaxons notent comme nous $[a\,;b]$ les intervalles fermés, pour les intervalles ouverts, ils notent $(a;b)$ là où notre habitude est d'utiliser $]a;b[$ ([2]).

Il n'a donc pas été prévu que des crochets tournés vers l'extérieur puissent être utilisés comme délimiteurs, et il en résulte des espacements désastreux si nous tapons p. ex. simplement `$x\in]0;1[$` ou `$[0;1]\cup]2;3]\cup[4;5[$` : $x \in]0;1[$ ou $[0;1]\cup]2;3] \cup [4;5[$. Pour résoudre une fois pour toutes ces problèmes, et en même temps pour que la taille des crochets s'adapte à la taille des bornes de l'intervalle, nous définissons comme suit quatre macros à deux arguments :

```
\newcommand{\cc}[2]{\left[#1;#2\right]}
\newcommand{\co}[2]{\left[#1;#2\right[}
\newcommand{\oc}[2]{\left]#1;#2\right]}
\newcommand{\oo}[2]{\left]#1;#2\right[}
```

(avec `c` pour *closed* et `o` pour *open*). Maintenant, `$x\in\oo01$` et `$\cc01\cup\oc23\cup\co45$` donnent $x \in]0;1[$ et $[0\,;1\,] \cup]2;3] \cup [4;5[$, avec des espacements corrects.

Dans une prochaine rubrique, nous décrirons une petite collection de macros utiles.

Pascal Dupont est chargé de cours à HEC•ULg. ✉ `pascal.dupont@ulg.ac.be`.

([2]) Donald Knuth qualifie d'ailleurs de « pervers » les mathématiciens qui utilisent les crochets à l'envers (*The TEXbook*, p. 171, exercice 18.14.)