Karnaj

Zeste de savoir

La TeXnique : Les commandes de définition

23 avril 2020

Table des matières

En bon utilisateur de LaTeX, on est habitué à voir `\newcommand` utilisé pour définir de nouvelles commandes. Et après s'être un peu renseigné sur TeX, on apprend que `\newcommand` est construite sur la base de la primitive `\def`. On obtient alors un outil plus simple à utiliser que `\def` et aussi plus sécurisé.

1. Les commandes de définition

La commande `\def` est assez connue, mais il y en a d'autres que nous allons voir ici. J'ai par exemple déjà parlé de `\edef` dans un précédent billet .

En fait, la définition de commandes se place sur deux axes. Celui de l'expansion (est-ce que la commande de définition développe directement son argument) et celui de la localité (est-ce que la commande définie existe dans tout le reste du document ou juste dans le groupe où elle est définie). On a alors ce tableau de commandes.

	Développe son argument	**Ne développe pas son argument**
Définit des commandes locales	`edef` (pour *expanded definition*)	`def`
Définit des commandes globales	`xdef`	`gdef` pour (*global definition*)

TABLE 1.2. – Les commandes de définition.

Faisons quelques tests pour vérifier tout ça.

```
```

Listing 1 – Test de la globalité des commandes définies.

Ici, si un appel à `\d` ou à `\ed` en dehors du bloc où ils sont définis (il suffit de supprimer les commentaires et de les remplacer par l'appel pour s'en rendre compte) alors que les appels à `\gd` et à `\xd` ne posent aucun problème. `\d` et `\ed` existent seulement dans le bloc où ils sont définis alors que `\xd` et `\ed` existent aussi en dehors.

Pour tester dans quel cas l'argument de la commande est développé, nous allons reprendre cet

exemple du billet sur l'expansion.

Avec `\def` et `\gdef`, nous obtenons « z » : le `a\a` de la définition de `\aa` n'a pas été développée et donc lors de l'appel de `\aa`, c'est la nouvelle commande `\a` (qui vaut `z`) qui est utilisée. Avec `\edef` et `\xdef`, nous obtenons « a » : le `a\a` de la définition de `\aa` a cette fois été développé directement en `aa`, dès la définition de la commande.

Dans la suite, nous allons travailler majoritairement avec `\def`, mais les informations restent vraies pour les autres commandes. Les seules différences entre ces commandes de définition sont celles que nous venons de voir.

2. Des commandes à paramètres

Pour le moment, nous n'avons défini que des commandes sans argument. Voyons une première commande avec des arguments.

Ici, nous définissons une commande à deux arguments. La forme du `\def` est sensiblement différente de celle de `\newcommand`. On ne donne pas le nombre d'arguments, mais on écrit directement chaque `#<n>` après le nom de la commande. Une commande peut donc avoir jusqu'à neuf paramètres (de `#1` à `#9`).

Numérotation des paramètres

Les paramètres doivent apparaître dans l'ordre croissant et doivent être des entiers consécutifs. Sinon nous aurons l'erreur *Parameters must be numbered consecutively.*

Ici, nous obtenons une erreur car les paramètres demandés, `#2#1` ne sont pas dans l'ordre croissant.

Ceci nous permet de comprendre qu'une définition a 4 parties :

- la commande de définition ;
- le nom de la commande à définir ;
- le texte de paramètres ;
- le texte de remplacement.

La commande de définition correspond à l'une des commandes que nous avons vues précédemment, le nom de la commande à définir... Je pense que nous savons à quoi il correspond.
Le texte de remplacement vient en dernier, entre accolades, après le texte de remplacement. Il correspond au modèle utilisé par la commande et peut utiliser les différents paramètres définis grâce au texte de paramètres. Dans notre commande `citation`, on a un modèle qui affiche le premier paramètre entre guillemets, puis le second paramètre après `--`.
Et finalement, le texte de paramètres est le `#1#2` que nous le voyons dans la commande `citation`. Il comprend la liste des paramètres que prend la commande. À première vue, c'est donc

une liste de `#<n>` avec les entiers `n` croissants et consécutifs comme nous l'avons vu plus haut. Mais nous allons voir qu'il peut être bien plus.

Lorsque nous créons une commande à l'intérieur d'une commande, les arguments de la commande interne s'écrivent avec deux `##` (il faut pouvoir les distinguer de ceux de la commande externe).

3. Les paramètres délimités

Si le texte de paramètre est si puissant, c'est qu'il existe deux types de paramètres. Les paramètres non délimité sont ceux que nous avons jusqu'à présent. Dans la définition d'une commande qui n'a que des paramètres non délimités, ils apparaissent l'un après l'autre. Il n'y a rien entre eux et rien entre le dernier et le texte de remplacement. Et elle s'utilise en mettant chaque argument après l'appel de la commande, chaque argument étant soit un *token*, soit un groupe de *token* entre accolades.
Ainsi, `\citation{a}{b} c` correspond à `\citation a b c`, à `\citation {a} b` ou encore à `\citation a {b} c` et les deux arguments sont `a` et `b`.
Avec les paramètres délimités, on peut délimiter les paramètres de manière plus fine, de manière à respecter un modèle. En gros, les paramètres délimités sont délimités par une liste de *tokens*. C'est cette liste de *tokens* qui indique à LaTeX qu'il a fini de lire ce qu'on voulait donner en argument. Quand LaTeX cherche un argument correspondant à un paramètre délimité, il prend donc tout, jusqu'à trouver la liste de *tokens* qui le délimite.
Les délimiteurs s'insèrent dans le texte de paramètres, et le texte de paramètres correspond alors au modèle qu'on doit respecter. En voici un exemple.

Ici, pour utiliser `\citation`, le nom de la commande doit être suivi du premier argument entre guillemets (avec des espaces), suivi du second argument après `de` (là encore les espaces sont importants, puis un point.

Les espaces

Puisqu'une suite d'espaces est interprétée comme une seule espace, les modèles `« #1 » de #2.` et `« #1 » de #2.` sont équivalents et lors de l'utilisation de la commande, il faut juste en mettre un nombre non nul pour respecter le modèle (c'est aussi ce qui permettait d'écrire `\citation a b c` plus haut).

Notons que les délimiteurs ne peuvent pas être `#` et `{`, puisque `#` indique le prochain paramètre et que `{` indique le début du texte de remplacement. En fait, un paramètre est non délimité s'il est suivi de `#` ou de `{` (il n'y a pas de délimiteur entre lui et le paramètre suivant ou le texte de remplacement).
De plus, nous pouvons très bien avoir les deux types de paramètres dans une commande.

```
```

Ici, on a délimité le premier paramètre de `\citation` grâce à `de`. Le second paramètre ne l'est pas et pour que TeX comprenne qu'il s'agit de `Clem`, nous avons dû le placer entre accolades. Sans ça, il n'aurait considéré que le `C`. Et c'est le second paramètres de `\citationAux` qui est délimité (par un point) là où le premier ne l'est pas. Si nous n'avins pas placé la citation entre accolades avec `\citationAux`, le premier paramètre aurait été `Z` et le second tout ce qui suit jusqu'au point.

C'est la fin de ce billet. Les commandes de définition de TeX sont plus « puissantes » que `\newcommand`, mais elles sont quand même moins pratiques à écrire, non ? Et puis les *packages* comme `xparse` sont là pour aider à créer des commandes en LaTeX (je parle de `xparse` dans quasiment tous les billets, j'ai l'impression de faire de la pub) et n'oublions pas que nous pouvons programmer en Lua avec LuaTeX.
Voilà, ça fait un bout de temps depuis le dernier billet alors que j'avais déjà le thème et le plan de ce billet (et de celui qui suit) quand je terminais d'écrire le précédent.