Non-tuto LaTeX

Alice M.
www.alicem.net

Version du 27 juin 2017

Introduction

Ce document n'a pas vocation à être un tutoriel LaTeX : il en existe déjà plein, dont des français, et il y a aussi des wikibooks qui, bien qu'ils soient (comme toujours avec LaTeX) incomplets, sont tout de même assez utile. Non, je veux ici juste rassembler des trucs que j'aurais bien aimé qu'on me dise ; le genre de choses qu'on trouve parfois dans un recoin d'un commentaire d'une réponse à une question sur un site web, et qui aurait pu nous faire gagner des heures si on l'avait vu plus tôt. C'est aussi l'occasion de partager des trucs que je trouve sympas, et de montrer que certains aspects à première vue obscurs de LaTeX ne sont en fin de compte pas si bordéliques et compliqués que ça. C'est aussi une bonne occasion pour moi de réviser un peu. Et puis, c'est marrant.

Avant de vous plonger dans ce document, veillez à au moins avoir de petites bases : comment pondre un document tout simple qui puisse être compilé sans erreur, comment créer une commande…

Notez que ce document peut évoluer avec le temps, au fil de mes découvertes et élans de motivation.

Table des matières

1 Gestion des espaces et coupure des lignes

Je suis souvent horrifié par la façon qu'ont les gens, et mêmes certaines personnes balèzes sur le net, de mettre en forme leurs commandes, ou plus généralement leur code. Rappelons donc juste deux trois trucs vite fait :

— Les espaces au début d'une ligne de code sont ignorés, ce qui vous permet d'indenter votre code sans perturber le rendu.

— Les espaces après une commande écrite sans accolades sont bouffés sans conséquences, donc vous pouvez remplacer :
`\bfseries\color{red}\itshape\hfill\strut\LaTeX\quadPlop.`
par :
`\bfseries \color{red}\itshape \hfill \strut \LaTeX \quad Plop.`

— Les espaces, ça rend le code plus lisible, mais parfois ça ne suffit pas. Sachez que vous pouvez revenir à la ligne à plus ou moins n'importe quel point du code si vous finissez votre ligne par un `%`. Je donnerai un exemple tout à l'heure.

— J'ai déjà vu des gens écrire `\macommande \space` pour contrecarrer le fait que LaTeX bouffe les espaces après `\macommande`. Or, il suffit bien souvent d'écrire `\macommande{}` pour rétablir un espacement qui respecte les règles de la langue choisie (via `babel`). Vous pouvez aussi charger le package `xspace` et ajouter `\xspace` à la fin de la définition de `\macommande` pour obtenir un effet similaire sans avoir à mettre les `{}` par la suite. Ainsi, ma commande personnelle « points de suspension » est définie ainsi :
`\newcommand{\ppp}{\ldots{}\xspace}`
J'ai ainsi un rendu correct sans `{}` avec la plupart des cas d'utilisation :
`A\ppp B.` ; `A (B\ppp) C.` ; `\fbox{A\ppp}`

— En mode maths, l'espacement entre la plupart des caractères est géré automatiquement. De ce fait, il y a plein d'endroits où vous pouvez mettre des espaces (voire des retours à la ligne) dans votre code pour le rendre plus lisible.

```
La commande suivante:

\newcommand{\moche}[1]{\medskip\noindent\fbox{\
     colorbox{gray}{\begin{minipage}{0.5\
     linewidth}\centering\bfseries\large#1\end{
     minipage}a}b}c\medskip}

\moche{Test}

peut également s'écrire ainsi:

\newcommand{\beau}[1]{
    \medskip

    \noindent
    \fbox{%
        \colorbox{gray}{%
            \begin{minipage}{0.5\linewidth}
                \centering
                \bfseries
                \large
                #1
            \end{minipage}%
            a%
        }%
        b%
    }%
    c

    \medskip
}

\beau{Test}

et le rendu est le même, bordel.

En revanche, si on coupe des lignes sans \com{\%
     }, on obtient des espaces, qui risquent d'
     apparaître dans le rendu s'ils ne sont pas
     au début d'une ligne:

\fbox{
    a
}
\rule{4em}{0.5em}
\colorbox{gray}{
    b
}

\fbox{%
    a%
}%
\rule{4em}{0.5em}%
\colorbox{gray}{%
    b%
}%

Donc mettez votre code en forme, mais pas sans ré
     fléchir.
```

La commande suivante :

peut également s'écrire ainsi :

et le rendu est le même, bordel.
En revanche, si on coupe des lignes sans `%`, on obtient des espaces, qui risquent d'apparaître dans le rendu s'ils ne sont pas au début d'une ligne :

Donc mettez votre code en forme, mais pas sans réfléchir.

```
Le passage mathématique suivant:
\[(x-i)(x+i)=0\Leftrightarrow\begin{cases}x-i=0\\
      x+i=0\end{cases}\]

peut également s'écrire:
\[
    (x - i)(x + i) = 0 \Leftrightarrow
    \begin{cases}
        x - i = 0 \\
        x + i = 0
    \end{cases}
\]
```

Le passage mathématique suivant :

$$(x-i)(x+i)=0\Leftrightarrow\begin{cases}x-i=0\\x+i=0\end{cases}$$

peut également s'écrire :

$$(x-i)(x+i)=0\Leftrightarrow\begin{cases}x-i=0\\x+i=0\end{cases}$$

2 Longueurs

J'ai mis un temps fou à utiliser les longueurs, et je vous déconseille de faire la même erreur. Si vous commencez à écrire plusieurs fois un truc genre `2cm` et que vous voulez vous assurer que cette distance reste la même à plusieurs endroits, donnez un nom à cette longueur et utilisez le nom. Ce n'est pas si compliqué que ça.

```
\newlength{\malongueur}
\setlength{\malongueur}{2em}

Rond.\hspace{\malongueur}Patate.

Rond.\hspace{\malongueur}Tulipe.
```

Rond. Patate.
Rond. Tulipe.

Certaines longueurs existent déjà dans LaTeX (définies ou ajustées selon la classe de document choisie, dans certains cas). Parmi celles que j'utilise le plus : `\baselineskip` est (en simplifiant un peu) la distance entre la base de deux lignes successives d'un paragraphe, `\linewidth` donne la largeur d'une ligne dans le contexte courant, `\parskip` la distance ajoutée entre deux paragraphes, `\parindent` l'alinéa, `\tabcolsep` l'espace entre colonnes dans un environnement `tabular`, `\fboxsep` pour l'espace interne de certaines boites... Note sur l'exemple suivant : j'utilise une commande maison débile `\pangr` pour pondre un pangramme et faire du remplissage. `\lipsum` est un peu long pour certains usages.

```latex
\begin{tabular}{|r|l|}
    \hline
    Tulipe & Fleur  \\
    Rond   & Rigolo \\
    \hline
\end{tabular}

\setlength{\tabcolsep}{2em}

\begin{tabular}{|r|l|}
    \hline
    Tulipe & Fleur  \\
    Rond   & Rigolo \\
    \hline
\end{tabular}

\pangr

\pangr

\setlength{\parskip}{2em}

\pangr
```

Tulipe	Fleur
Rond	Rigolo

Tulipe	Fleur
Rond	Rigolo

Le vif zéphyr jubile sur les kumquats du clown gracieux.
Le vif zéphyr jubile sur les kumquats du clown gracieux.

Le vif zéphyr jubile sur les kumquats du clown gracieux.

3 Boites

La notion de boites est omniprésente en LaTeX. Parmi celles que l'on peut avoir à créer soi-même, je citerai notamment `\mbox`, dont la largeur suit celle du contenu et qui empêche les retours à la ligne (je m'en sers pour protéger certaines équations qui deviennent pénibles à lire quand elles sont découpées), `\makebox`, qui fonctionne un peu pareil mais avec deux arguments optionnels pour la largeur et l'alignement horizontal.

```latex
Voici une équation: $1 + 2 + 3 + 4 + 5 < 6 + 7 +
    8 + 9 + 10$. Vroum badaboum.

\medskip

Avec une \com{mbox}:\\
Voici une équation: \mbox{$1 + 2 + 3 + 4 + 5 < 6
    + 7 + 8 + 9 + 10$}. Vroum badaboum.

\medskip

Je me sers parfois de boites pour faire des \og
    marges contenant quelque chose qui se fout à
     droite \fg{}, comme pour faire des listes \
    og maison \fg{}:

\newcommand{\fauxitem}{%
    \makebox[3em][r]{%
        $\bullet$\hspace*{0.25em}%
    }%
}

\fauxitem Poire.\\
\fauxitem Banane.\\
\hspace*{3em}Ligne pour montrer que je contrôle
    la taille de la marge (le texte commence au
    même niveau que celui de la liste ci-dessus,
     sans compter les puces ($\bullet$)).
```

Voici une équation : $1 + 2 + 3 + 4 + 5 < 6 + 7 + 8 + 9 + 10$. Vroum badaboum.

Avec une `mbox` :
Voici une équation : $1 + 2 + 3 + 4 + 5 < 6 + 7 + 8 + 9 + 10$. Vroum badaboum.

Je me sers parfois de boites pour faire des « marges contenant quelque chose qui se fout à droite », comme pour faire des listes « maison » :

- Poire.
- Banane.

Ligne pour montrer que je contrôle la taille de la marge (le texte commence au même niveau que celui de la liste ci-dessus, sans compter les puces (•)).

Sur mon exemple d'équation, ça donne tout de même un rendu bizarre et des avertissements. Il vaut souvent mieux couper manuellement la ligne (`\\`) avant l'équation ou la centrer avec `\[...\]`. Mais dans de longs paragraphes contenant une multitude de petites formules, ça peut être sympa de les protéger (elles auront davantage de liberté de mouvement que dans mon exemple, et ça évitera d'avoir un rendu horrible). `\mbox` peut aussi être sympa pour empêcher les retours à la ligne dans un mot, genre un nom d'entreprise ou autre.

Il y a aussi des boites un peu plus tordues, pour contrôler la taille de ce qu'elles contiennent (définies par le package `graphicx`).

En gros, `\scalebox` fait un zoom avec un facteur en argument, tandis que `\resizebox` permet de « viser » des dimensions données, en utilisant éventuellement `!` pour une des deux dimensions si on veut la calculer automatiquement en conservant les proportions d'origine. Un usage classique de `\resizebox` est l'adaptation d'un tableau ou d'une figure à la largeur de la page (ou colonne, ou ligne...).

```
\newcommand{\contenu}{\LARGE\LaTeX}

\contenu%
\hfill%
\scalebox{2}{\contenu}%
\hfill%
\scalebox{0.5}{\contenu}

\resizebox{0.5\linewidth}{8pt}{\contenu}%
\hfill%
\resizebox{!}{8pt}{\contenu}%
\hfill%
\resizebox{0.25\linewidth}{!}{\contenu}%
```

Il existe aussi `\raisebox`, qui fait selon moi partie de ces trucs qui sont dégueulasses mais qui DOIVENT exister pour nous sauver la vie dans quelques rares cas critiques. `\raisebox` décale son contenu verticalement, et permet aussi (via les arguments optionnels) de mentir allègrement sur la hauteur et la profondeur (distance verticale sous la « ligne de base » du texte) de l'élément résultant.

```
\centering

a\raisebox{0.333em}{b}c%
\qquad%
a\raisebox{-0.333em}{b}c

\medskip

\newcommand{\truc}{%
    \colorbox{black}{%
        \begin{minipage}{4em}
            \centering
            \color{white}
            \bfseries
            Truc\\
            assez\\
            haut
        \end{minipage}%
    }%
}

En mentant sur la hauteur et la profondeur d'un
     gros élément, j'arrive à le faire déborder
     sur les lignes du dessus et d'en dessous:

\dotfill\\
\dotfill\raisebox{0pt}[0pt][0pt]{\truc}\dotfill\\
\dotfill

\medskip

À l'inverse, on peut réserver trop de place:

\dotfill\\
\dotfill\raisebox{0pt}[4em][3em]{\truc}\dotfill\\
\dotfill
```

a$^{\text{b}}$c a$_{\text{b}}$c

En mentant sur la hauteur et la profondeur d'un gros élément, j'arrive à le faire déborder sur les lignes du dessus et d'en dessous :

..... **Truc** .....
..... **assez** .....
..... **haut** .....

À l'inverse, on peut réserver trop de place :

.....

..... **Truc** **assez** **haut** .....

.....

4 Indentation des paragraphes

`\parindent` mérite qu'on s'attarde un peu sur lui. Cette distance est généralement ajoutée par défaut en espace horizontal aux débuts des nouveaux paragraphes. On peut empêcher ponctuellement ça avec un `\noindent`, on globalement en mettant `\parindent` à zéro (`0pt`, `0cm`...). Notez qu'au sein d'une `minipage` (je parlerai probablement un peu plus des `minipage`s plus tard), `\parindent` est mis à zéro (à vrai dire, c'est souvent ce que l'on souhaite dans ce contexte). Pour rétablir l'indentation qu'on avait avant d'ouvrir la `minipage`, il peut être bon d'enregistrer la valeur au préalable.

```latex
\setlength{\parindent}{2em}

\newlength{\sauvegardeparindent}
\setlength{\sauvegardeparindent}{\parindent}

\centering

\begin{minipage}{\linewidth}
    1: Pas indenté car dans minipage.
\end{minipage}

\bigskip

\begin{minipage}{\linewidth}
    \setlength{\parindent}{\sauvegardeparindent}
    2: Indenté car récupération de la valeur de \
        com{\bs parindent}.

    \noindent
    3: Pas indenté car \com{\bs noindent}.
\end{minipage}
```

1 : Pas indenté car dans minipage.

2 : Indenté car récupération de la valeur de `\parindent`.
3 : Pas indenté car `\noindent`.

Un truc assez fréquent est de se bouffer un avertissement (en plus d'un rendu moche) à la compilation pour une « overfull hbox » suite à un oubli de `\noindent` ou de `\centering`, notamment en voulant créer une boîte prenant toute la largeur de la ligne.

```latex
\setlength{\parindent}{2em}

\parbox{\linewidth}{Déborde car indenté!\dotfill
    |}

\noindent
\parbox{\linewidth}{OK (\com{\bs noindent})\
    dotfill|}

\centering
\parbox{\linewidth}{OK (centré)\dotfill|}
```

Déborde car indenté !.....|
OK (`\noindent`).....|
OK (centré).....|

5 Microtypographie et césures

Je me dois de faire un peu de pub au package `microtype`, que j'inclue maintenant presque toujours, sans même lui donner d'option, et qui fait des miracles. En gros, ça améliore principalement la répartition du texte pour mieux justifier les paragraphes, etc. J'ai des textes (surtout sur du papier A5, d'ailleurs, car lignes plus courtes) qui sont passés de « un avertissement pour *overfull hbox* toutes les deux pages » à « aucun avertissement » juste en incluant ce package. Après, si vous voulez vous plonger dans la documentation très fournie (`texdoc microtype`), libre à vous, mais bon. Si vous avez encore des avertissements, vous pouvez régler la longueur `\emergencystretch` (qui, en fait, ne vient pas de `microtype` mais est présente de base) pour donner un peu plus de liberté aux algorithmes de césure (coupage de ligne) et compagnie. Perso, je la mets à `5pt` en cas de problème. Mais pensez aussi à vérifier que votre problème ne vient pas simplement d'un mot peu usuel que LaTeX ne sait pas comment découper (voire carrément d'une erreur de mise en page genre l'oubli de `\noindent` mentionné plus tôt) ! Les mots composés peuvent typiquement poser problème, alors n'hésitez pas à suggérer des points de césure avec `\-`.

```
Badaboudoudoum palapouloupoum voiture-ananas.

Badaboudoudoum palapouloupoum voi\-ture-ana\-nas.
```

Badaboudoudoum palapouloupoum voiture-ananas.
Badaboudoudoum palapouloupoum voi-
ture-ananas.

Si vous voulez justifier du texte sans qu'il y ait de césures (typiquement, pour un CV ou un diaporama), vous pouvez attribuer une pénalité exagérée au fait d'en mettre (`\hyphenpenalty=100000`) ou (et c'est probablement plus propre, en tout cas si vous voulez interdire les césures dans tout le document) faire `\usepackage[none]{hyphenat}`.

Certains ont tendance à abuser de `\sloppy` et de l'environnement `sloppypar` pour se débarrasser des avertissements et se fichent du rendu final. Personnellement, j'ai l'impression que si on suit les conseils que je viens de donner et qu'on ne présente pas son document de manière farfelue, ces choses bourrines ne sont quasiment jamais nécessaires, donc je n'en parlerai pas.

6 Minipages et parboxes

À mes yeux, les `minipage`s et les `parbox`es ont trois principaux intérêts : disposer des choses côte à côte, contrôler la largeur occupée par des éléments, et refiler à une commande quelque chose de plus complexe que ce qu'on est a priori censé lui refiler.

Les différences entre `minipage`s et `parbox`es sont assez minimes. Une `minipage` va réinitialiser le compteur des listes, absorber les notes de pied de page (un sujet discuté sur quarante-mille sites), et s'aligner un peu plus facilement dans certains contextes. Mais bon. Perso, ma règle est « si je dois empaqueter des choses vraiment bordéliques, j'utilise une `minipage`, et s'il ne s'agit que d'un ou deux paragraphes j'utilise une `parbox`.

```
\begin{minipage}{0.25\linewidth}
    [\dotfill1\dotfill]
\end{minipage}%
\begin{minipage}{8em}
    [\dotfill2\dotfill]
\end{minipage}

Sans le \com{\%} à la fin de la première \com{
    minipage}, on se retrouve avec un espace
    entre les deux, ce qui peut être ou non dé
    sirable:

\begin{minipage}{0.25\linewidth}
    [\dotfill1\dotfill]
\end{minipage}
\begin{minipage}{8em}
    [\dotfill2\dotfill]
\end{minipage}

\parbox{0.4\linewidth}{%
    \raggedright
    \pangr
}%
\parbox{0.5\linewidth}{%
    \raggedright
    \pangrlong
}
```

[. . . . 1 . . . .][. . . . . 2 . . . . .]
Sans le `%` à la fin de la première `minipage`, on se retrouve avec un espace entre les deux, ce qui peut être ou non désirable :
[. . . . 1 . . . .] [. . . . . 2 . . . . .]

Le vif zéphyr jubile sur les kumquats du clown gracieux.

Portez ce vieux whisky au juge blond qui fume sur son île intérieure, à côté de l'alcôve ovoïde, où les bûches se consument dans l'âtre, ce qui lui permet de penser à la cænogénèse de l'être dont il est question dans la cause ambiguë entendue à Moÿ, dans un capharnaüm qui, pense-t-il, diminue çà et là la qualité de son œuvre.

Une `minipage` ou une `parbox` (ou un peu n'importe quelle boite, en fait, je crois) est interprétée comme un unique gros caractère. Imaginez un « A » gigantesque : il ne sera pas coupé entre deux pages, et pourra éventuellement se ranger (quoiqu'un peu maladroitement) sur la ligne courante. Un intérêt majeur de cela est que quand une commande s'attend à recevoir du texte tout bête, vous pouvez lui donner du bordel si vous avez empaqueté ce bordel dans une `minipage` ou autre.

```
Les \com{minipage}s, de même que \com{tabular},
    permettent de revenir à la ligne dans un nœ
    ud TikZ (on peut aussi utiliser l'attribut \
    com{align} du nœud):

\begin{tikzpicture}
    \node[circle, draw = red] (0, 0) {%
        \begin{minipage}{7em}
            \centering
            Pomme\\
            Poire tourte
        \end{minipage}
    };
\end{tikzpicture}

\medskip

Idem pour tenter la rotation de choses complexes:

\rotatebox{45}{%
    \begin{minipage}{7em}
        \centering
        Pomme\\
        \dotfill\\
        Poire tourte
    \end{minipage}%
}
```

Les `minipage`s, de même que `tabular`, permettent de revenir à la ligne dans un nœud TikZ (on peut aussi utiliser l'attribut `align` du nœud) :

Idem pour tenter la rotation de choses complexes :

Les `minipage`s et `parbox`es ont des paramètres pour gérer l'alignement et ce genre de trucs. Les `minipage`s ont trois arguments optionnels (mais plein de sites ne parlent que du premier ! C'est honteux) : l'alignement par rapport au contexte extérieur, la hauteur, et la position du contenu de la `minipage`.

```
(Les \com{fbox}es sont juste là pour vous montrer
      les contours des \com{minipage}s.)

\medskip

[Extérieur]
\fbox{%
    \begin{minipage}[t]{0.65\linewidth}
        \raggedright
        \com{t} (\og top \fg{}): Première ligne
             de la \com{minipage} alignée avec la
              ligne extérieure.
    \end{minipage}%
}

\medskip

[Extérieur]
\fbox{%
    \begin{minipage}[c]{0.65\linewidth}
        \raggedright
        \com{c} (\og center \fg{}): Centre de la
             \com{minipage} alignée avec la ligne
              extérieure.
    \end{minipage}%
}

\medskip

[Extérieur]
\fbox{%
    \begin{minipage}[b]{0.65\linewidth}
        \raggedright
        \com{b} (\og bottom \fg{}): Dernière
             ligne de la \com{minipage} alignée
             avec la ligne extérieure.
    \end{minipage}%
}
```

(Les `fbox` es sont juste là pour vous montrer les contours des `minipage` s.)

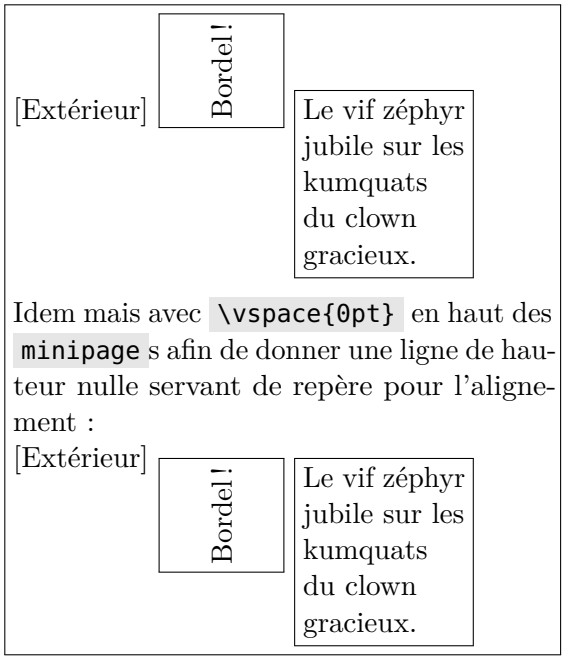
[Extérieur] `t` (« top ») : Première ligne de la `minipage` alignée avec la ligne extérieure.

[Extérieur] `c` (« center ») : Centre de la `minipage` alignée avec la ligne extérieure.

[Extérieur] `b` (« bottom ») : Dernière ligne de la `minipage` alignée avec la ligne extérieure.

Pour aligner proprement des `minipage` s contenant des choses un peu plus complexes que des lignes de texte, il peut être nécessaire d'utiliser des techniques perdues au milieu de la documentation de `epslatex` (qui n'a rien à voir avec ça à la base), afin de créer des genres de lignes invisibles sur lesquelles se feront l'alignement.

```
[Extérieur]
\fbox{%
    \begin{minipage}[t]{0.2\linewidth}
        \centering
        \rotatebox{90}{Bordel!}
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[t]{0.3\linewidth}
        \raggedright
        \pangr
    \end{minipage}%
}

\medskip

Idem mais avec \com{\bs vspace\{0pt\}} en haut
     des \com{minipage}s afin de donner une ligne
      de hauteur nulle servant de repère pour l'
     alignement:

[Extérieur]
\fbox{%
    \begin{minipage}[t]{0.2\linewidth}
        \vspace{0pt}

        \centering
        \rotatebox{90}{Bordel!}
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[t]{0.3\linewidth}
        \vspace{0pt}

        \raggedright
        \pangr
    \end{minipage}%
}
```



```
Mais bon, après c'est aligné avec le haut de la \
    com{minipage} plutôt qu'avec le haut de son
    contenu, donc ça peut être un peu foireux.
    Mais en général quand on en arrive là c'est
    qu'on est pas à ça près. Dans certains cas,
    vous pouvez mettre l'extérieur dans sa
    propre \com{minipage} pour arranger ça.
    Sinon, c'est probablement que vous êtes en
    train de faire un truc un peu sale.

\fbox{%
    \begin{minipage}[t]{0.25\linewidth}
        \vspace{0pt}

        [Extérieur]
    \end{minipage}
}
\fbox{%
    \begin{minipage}[t]{0.2\linewidth}
        \vspace{0pt}

        \centering
        \rotatebox{90}{Bordel!}
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[t]{0.3\linewidth}
        \vspace{0pt}

        \raggedright
        \pangr
    \end{minipage}%
}
```

Mais bon, après c'est aligné avec le haut de la `minipage` plutôt qu'avec le haut de son contenu, donc ça peut être un peu foireux. Mais en général quand on en arrive là c'est qu'on est pas à ça près. Dans certains cas, vous pouvez mettre l'extérieur dans sa propre `minipage` pour arranger ça. Sinon, c'est probablement que vous êtes en train de faire un truc un peu sale.

[Extérieur]

Bordel!

Le vif zéphyr jubile sur les kumquats du clown gracieux.

```
L'astuce du \com{\bs vspace\{0pt\}} que l'on
     vient de voir marche aussi avec le
     positionnement \com{b}, en mettant le \com{\
     bs vspace\{0pt\}} à la fin:

\medskip

\textbf{Sans:}

[Extérieur]
\fbox{%
    \begin{minipage}[b]{0.2\linewidth}
        \centering
        \rotatebox{90}{Bordel!}
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[b]{0.3\linewidth}
        \raggedright
        \pangr
    \end{minipage}%
}

\medskip

\textbf{Avec:}

[Extérieur]
\fbox{%
    \begin{minipage}[b]{0.2\linewidth}
        \centering
        \rotatebox{90}{Bordel!}

        \vspace{0pt}
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[b]{0.3\linewidth}
        \raggedright
        \pangr

        \vspace{0pt}
    \end{minipage}%
}
```

L'astuce du `\vspace{0pt}` que l'on vient de voir marche aussi avec le positionnement `b`, en mettant le `\vspace{0pt}` à la fin :

Sans :

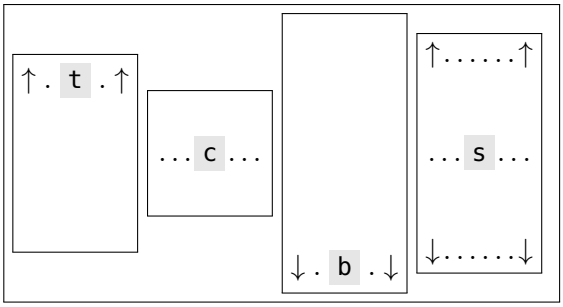
[Extérieur] Bordel! Le vif zéphyr jubile sur les kumquats du clown gracieux.

Avec :

[Extérieur] Bordel! Le vif zéphyr jubile sur les kumquats du clown gracieux.

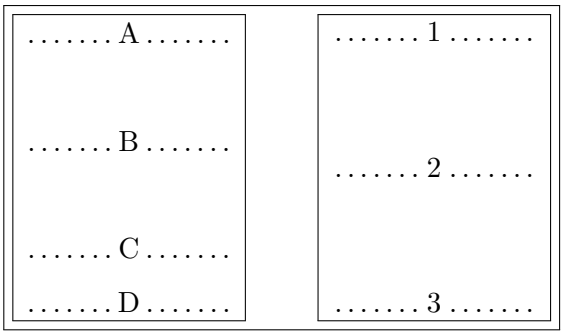
Voyons maintenant les autres paramètres optionnels de `minipage` : la hauteur et la position du contenu. C'est un peu plus simple. Notez la présence d'une valeur `s` (« stretch ») pour la position du contenu, qui étire tout espace vertical traînant dans la boite (j'ai mis des `\vfill`) afin de répartir le contenu.

```latex
\fbox{%
    \begin{minipage}[c][5\baselineskip][t]{0.2\
        linewidth}
        $\uparrow$\dotfill\com{t}\dotfill$\
            uparrow$
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[c][3\baselineskip][c]{0.2\
        linewidth}
        \dotfill\com{c}\dotfill
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[c][7\baselineskip][b]{0.2\
        linewidth}
        $\downarrow$\dotfill\com{b}\dotfill$\
            downarrow$
    \end{minipage}%
}
\fbox{%
    \begin{minipage}[c][6\baselineskip][s]{0.2\
        linewidth}
        $\uparrow$\dotfill$\uparrow$

        \vfill

        \dotfill\com{s}\dotfill

        \vfill

        $\downarrow$\dotfill$\downarrow$
    \end{minipage}%
}
```



Un truc que j'aime beaucoup pour les posters, CV et diaporamas est de mettre côte à côte des `minipage`s de même hauteur et de répartir leur contenu avec de magnifiques `\vspace{\stretch{<n>}}`, où `<n>` est un coefficient d'étirement. Un `\stretch{2}` bouffera quatre fois plus d'espace qu'un `\stretch{0.5}`, par exemple. Ces commandes font du remplissage vertical. On peut aussi combiner ça avec un `\hfill` pour faire du remplissage horizontal entre les `minipage`s.

```latex
\fbox{%
    \begin{minipage}[c][8\baselineskip]{0.4\
         linewidth}
        \dotfill A\dotfill

        \vspace{\stretch{1}}

        \dotfill B\dotfill

        \vspace{\stretch{1}}

        \dotfill C\dotfill

        \vspace{\stretch{0.25}}

        \dotfill D\dotfill
    \end{minipage}%
}%
\hfill%
\fbox{%
    \begin{minipage}[c][8\baselineskip]{0.4\
         linewidth}
        \dotfill 1\dotfill

        \vspace{\stretch{1}}

        \dotfill 2\dotfill

        \vspace{\stretch{1}}

        \dotfill 3\dotfill
    \end{minipage}%
}%
```



7 Texte en maths

Une erreur que je vois souvent consiste à écrire sans se poser de question plusieurs lettres à la suite en mode maths. Il faut bien comprendre qu'un truc genre `$plop$` sera interprété comme « le produit de p, l, o et encore p », et pas du tout comme un nom de variable ou autre. Ça peut sembler anodin, mais le crénage – l'écart entre les lettres, etc. – va en prendre un coup. Dans l'exemple qui suit, j'ai utilisé des mots contenant « ff », car cela met particulièrement bien en évidence ce problème.

```latex
$x + affiche$ NON

$x + \mathit{affiche}$ OUI

$afficher(x)$ NON

$\operatorname{afficher}(x)$ OUI

% Dans le préambule :
% \DeclareMathOperator{\afficher}{afficher}
$\afficher(x)$ OUI

$\{x \in \mathbb{R} \mid x est pair et x \neq
     2\}$ NON

$\{x \in \mathbb{R} \mid x \text{ est pair et } x
      \neq 2\}$ OUI
```

$x + affiche$ NON
$x + \mathit{affiche}$ OUI
$afficher(x)$ NON
$\operatorname{afficher}(x)$ OUI
$\afficher(x)$ OUI
$\{x \in \mathbb{R} \mid x est pair et x \neq 2\}$ NON
$\{x \in \mathbb{R} \mid x \text{ est pair et } x \neq 2\}$ OUI

Pour écrire le nom d'une fonction contenant plusieurs lettres, la marche à suivre est selon moi celle-ci :

1. Voir si la commande correspondante existe déjà. En effet, `\cos`, `\sin`, `\max`, `\exp`, et pas mal d'autres sans doute existent déjà et permettent d'avoir un rendu cohérent. Tapez le nom de commande qui vous semble logique, et voyez si ça marche.
2. Si elle n'existe pas, demandez-vous si vous en aurez besoin une seule fois ou plusieurs. Pour un usage ponctuel, un `\operatorname{plop}` peut suffir.
3. Si vous comptez mentionner votre fonction à de nombreuses reprises, il vaut probablement mieux déclarer son existence via `\DeclareMathOperator{\commande}{texte}`. Cette commande est fournie par `amsmath`, si je ne me trompe pas. Dans cet exemple, cela déclare `\commande`, un opérateur qui affiche le texte « texte ».

Pour afficher des bouts de textes dans des formules, pensez également à :
— `\mathrm` : $a^2 + \mathrm{texte} - \sqrt{x}$;
— `\mathit` : $a^2 + \mathit{texte} - \sqrt{x}$ (je l'utilise surtout pour les noms de variables, y compris ceux sous forme d'acronymes du style UB. Comparez avec `$UB$` : $UB \neq UB$) ;
— `\mathsf` : $a^2 + \mathsf{texte} - \sqrt{x}$;
— `\mathbf` : $a^2 + \mathbf{texte} - \sqrt{x}$;
— `\text` : $a^2 + \text{texte} - \sqrt{x}$.

Il existe aussi l'alphabet calligraphique, avec `\mathcal`. Il n'est clairement pas conçu pour écrire des mots, ne serait-ce que parce qu'il ne contient généralement que les lettre capitales ($\mathcal{POIRE}$), mais on peut parfois avoir tout de même besoin d'enchaîner des lettres dans ce style… J'ai par exemple pas mal écrit « $\mathcal{NP}$-complet », mais je trouvais que ce « $\mathcal{NP}$ » était fichtrement large. Dans de tels cas, il peut être intéressant, en plus de bien entendu mettre tout ça dans une commande, de faire un peu de crénage manuel :

$\mathcal{NP}$ (`$\mathcal{NP}$`)
$\mathcal{NP}$ (`$\mathcal{N\kern -0.125em P}$`)

8 Groupes

Un couple d'accolades forme un *groupe*, de même que les commandes `\begingroup` et `\endgroup`, plus longues mais aussi plus visibles et explicites. Les groupes permettent de limiter des modifications à une zone du code.

```
\newcommand{\fruit}{Pomme}
\colorlet{couleur}{blue}

\textcolor{couleur}{1: \fruit}

\begingroup
    \renewcommand{\fruit}{Poire}
    \colorlet{couleur}{red}
    \textcolor{couleur}{2: \fruit}
\endgroup

\textcolor{couleur}{3: \fruit}
```

1 : Pomme
2 : Poire
3 : Pomme

Dans l'exemple qui précède, on constate que la redéfinition de la commande `\fruit` est perdue à la sortie du groupe, de même que la modification de la couleur `couleur`. Ce type de comportement peut être particulièrement intéressant pour faire des modifications

temporaires, ou ne pas polluer le « *scope* global » avec des commandes qui ne serviront que dans un petit coin du code ou dans une macro.

Notez aussi que les fins de groupes mettent fin aux commandes telles que `\itshape`, `\Large` ou `\color`, qui fonctionnent comme des interrupteurs et agissent jusqu'à la fin du groupe courant ou jusqu'à une directive contraire.

```
Pomme
{
    \itshape%
    \Large%
    \color{blue}%
    Poire
}
Banane
```

Pomme *Poire* Banane

Notez que certaines rares choses outrepassent les limites des groupes, comme par exemple les modification de valeur des compteurs. Retenez aussi qu'il existe des moyen de rendre global quelque chose qui n'aurait pas dû l'être.

```
\newcounter{num}

\newlength{\longa}
\setlength{\longa}{1cm}
\newlength{\longb}
\setlength{\longb}{2cm}

num: \arabic{num}\\
a: \rule{\longa}{2pt}\\
b: \rule{\longb}{2pt}

{
    \setcounter{num}{12}
    \setlength{\longa}{4cm}
    \setlength{\longb}{5cm}
    \global\longb=\longb
}
num: \arabic{num}\\
a: \rule{\longa}{2pt}\\
b: \rule{\longb}{2pt}
```

num : 0
a :
b :
num : 12
a :
b :

Enfin, sachez que pas mal de boîtes, d'environnement, etc. créent déjà des groupes. Ainsi, par exemple, un `\Large` dans un `itemize` ne fera effet que jusqu'à la fin de la liste, un `\centering` restera confiné à une figure ou à une minipage, etc.

Bien entendu, les groupes peuvent être imbriqués. Il y a une limite, mais en général quand on l'atteint c'est qu'on a vraiment cherché la merde.

9 Centrer

Ça peut sembler idiot de faire une section sur le fait de centrer du texte, mais il se trouve que c'est un sujet plus riche qu'on ne le croit souvent.

```latex
Pomme
\begin{center}
    Poire
\end{center}
Banane

\centerline{Framboise}
Citron

{%
    \centering%
    Fraise%
}

Melon

{%
    \centering%
    Pastèque\par
}
Orange
```

Pomme

Poire

Banane

Framboise

Citron
Fraise
Melon

Pastèque

Orange

Dans cette liste de fruits qui sert d'exemple, on peut principalement remarquer deux choses :

— L'environnement `center` ajoute de l'espace vertical avant et après lui, tandis que ce n'est pas le cas de `\centering` et `\centerline`. On peut retirer cet espace, mais ça revient un peu à s'embêter à retirer l'alcool d'une bière quand on peut boire du thé. Notez que cet espacement vertical a tendance à favoriser les sauts de page, ce qui peut être souhaitable ou non selon les cas.

— Un `\centering` ne fait effet sur un paragraphe que si on déclare explicitement que ce paragraphe est terminé et est prêt à être affiché, et si on déclare cela *avant* que le `\centering` ne cesse de faire effet. Dans l'exemple, on voit que « Fraise » n'est pas centré du tout. Cela vient du fait qu'au moment où le paragraphe que compose ce mot se termine (c'est-à-dire, quand on laisse une ligne vide avant « Melon », le `\centering` a déjà cessé de faire effet (il s'est arrêté à l'accolade fermante qui suit « Fraise »). À l'inverse, « Pastèque » est bien centré car un `\par` (équivalent à une ligne vide) a été placé juste après lui et avant de fermer le groupe contenant le `\centering`.

Cette histoire de valider un paragraphe avant que quelque chose ne cesse de faire effet est valable pour d'autres choses. Par exemple, cela peut être important pour avoir un interligne qui s'adapte à la taille de police. Voyez cela dans le petit exemple un peu hors-sujet qui suit.

```
Pomme

\begin{LARGE}
    Poire
\end{LARGE}

Tourte

\dotfill

Pomme

\begin{LARGE}
    Poire\par
\end{LARGE}

Tourte
```

Pomme
Poire
Tourte
. . . . . .
Pomme
Poire
Tourte

Notez d'ailleurs qu'à certains moments le paragraphe en cours est automatiquement « validé ». Inutile, par exemple, d'utiliser `\par` à la fin du contenu d'une minipage ou d'une parbox.

Mais revenons maintenant au centrage.

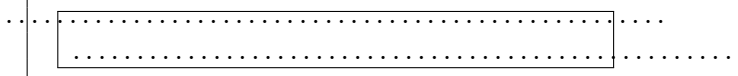
```
\hfill%
A%
\hfill%
B%
\hspace*{\fill}

\centerline{%
    C%
    \hfill%
    D%
}

\centerline{%
    E%
    \hspace{1cm}%
    F%
}
```

A B
C D
E F

Ce nouvel exemple montre que l'on peut utiliser du remplissage horizontal (`\hfill`) pour obtenir un genre de centrage, parfois avec une intéressante répartition de l'espace et du texte. La combinaison avec `\centerline`, par contre, n'est pas trop recommandée ; gardez `\centerline` pour les cas où vous avez des espaces fixes (voir la ligne avec le E et le F dans l'exemple).

Notez aussi que les espaces horizontaux peuvent rechigner à apparaître en fin de ligne, auquel cas il faut les imposer avec un `\hspace*`, en passant `\fill` en argument pour obtenir quelque chose d'équivalent à `\hfill`.

```
\centerline{%
    \makebox[1.25\linewidth]{%
        \dotfill%
    }%
}

{%
    \centering%
    \makebox[1.25\linewidth]{%
        \dotfill%
    }\par
}
```



Ce dernier exemple sur le sujet montre que `\centerline` peut être utilisé pour centrer des éléments de largeur un peu arbitraire (généralement sans avertissement à la compilation !), ce qui est assez pratique pour les tableaux, images, etc. Ceci ne fonctionne cependant pas avec `\centering`, et encore moins avec l'environnement `center`.

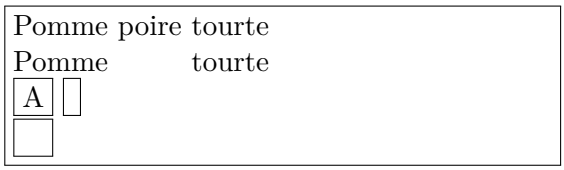
10 *Struts* et fantômes

Dans certains contextes, on se retrouve dépendant de la hauteur d'un bout de texte, voire de sa largeur. Pour avoir un rendu sympathique tout en évitant que le document ne devienne trop pénible à maintenir, il est bon de connaître quelques outils et de savoir lesquels employer selon la situation.

```
\fbox{aecox}
\fbox{jLgÏp}

\fbox{aecox\strut}
\fbox{jLgÏp\strut}
```

Dans ce petit exemple, des `\strut`s sont utilisés pour faire en sorte que deux boîtes aient la même hauteur. Un *strut* est un genre de bout de charpente qui pousse le sol et le plafond de manière à obtenir un truc d'une taille correspondant à ce qu'on peut obtenir avec des grosses lettres avec la police courante, aussi bien sous (*profondeur*) que sur (*hauteur*) la ligne de base. Concrètement, c'est une « règle de largeur nulle », un peu comme si on faisait «  », mais en invisible (`\rule{0pt}{3ex}`). D'ailleurs, on peut vraiment se servir de `\rule` pour créer ses propres *struts*, surtout si on utilise le paramètre optionnel de cette commande qui permet de décaler la règle obtenue verticalement et donc de lui donner de la profondeur : « ».

```
Pomme poire tourte

Pomme \hphantom{poire} tourte

\fbox{A} \fbox{\vphantom{A}}

\fbox{\phantom{A}}
```

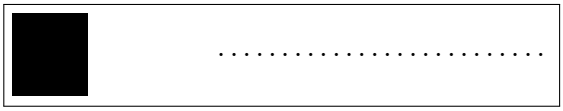


Cet exemple montre les effets de `\phantom`, `\hphantom` et `\vphantom`. Ils produisent respectivement du vide de la taille, de la largeur et de la hauteur de ce qu'on leur donne en argument.

11 Calculs de distances

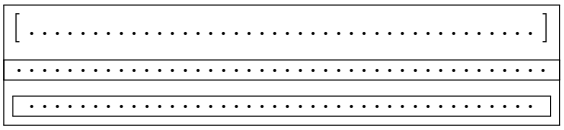
Le package `calc`, une fois inclus, ouvre sans vraiment d'efforts pas mal de possibilités pour effectuer des calculs avec des distances.

```
\begin{minipage}{0.3\linewidth - 1cm/2}
    \rule{1cm}{1cm - 2pt + 0.1\linewidth}
\end{minipage}%
\hspace{1cm}%
\begin{minipage}{0.7\linewidth - 1cm/2}
    \dotfill
\end{minipage}
```



C'est assez pratique pour, par exemple, calculer la largeur d'une seconde colonne en fonction de celle choisie pour la première, ou pour prendre en compte l'espacement interne (`\fboxsep`) et la largeur du trait (`\fboxrule`) des `\fbox`es et ainsi éviter un débordement :

```
[\dotfill]

\centerline{%
    \fbox{%
        \makebox[\linewidth]{\dotfill}%
    }%
}
\centerline{%
    \fbox{%
        \makebox[%
            \linewidth - 2\fboxsep - 2\fboxrule%
        ]{\dotfill}%
    }%
}
```



Certaines commandes, comme `\hspace`, semblent moins réceptives que d'autres à ces expressions calculatoires… En cas de besoin, vous pouvez toujours stocker le résultat dans une longueur temporaire et utiliser ensuite cette longueur nommée.

12 Formatage des nombres

Les packages `numprint` et `siunitx` permettent de mettre des nombres en forme sans se prendre la tête, et je pleins toujours les gens qui ne les connaissent pas, surtout ces chercheurs qui génèrent des tableaux à partir de vieilles données.

En général, ils prennent en compte les normes de la langue courante (mais pour `numprint`, mieux vaut passer l'option `autolanguage` lors de l'inclusion du package, et un `locale = FR` ne fait pas de mal pour `siunitx`), ce qui est plutôt cool. Il permettent aussi de gérer les quantités affublées d'unités avec les jolis espaces insécables qui vont bien.

```
Numprint: \np{1234.5678e-12}\par
Siunitx: \num{1234.5678e-12}

Numprint: \np[m^2]{12}\par
Siunitx: \SI{12}{m^2} ou \SI{12}{\square\meter}

Numprint: \np[\%]{50}\par
Siunitx: \SI{50}{\%} ou \SI{50}{\percent}
```

Numprint : $1\ 234{,}567\ 8 \cdot 10^{-12}$
Siunitx : $1234{,}5678 \times 10^{-12}$
Numprint : $12\,\mathrm{m}^2$
Siunitx : $12\,\mathrm{m}^2$ ou $12\,\mathrm{m}^2$
Numprint : 50 %
Siunitx : 50 % ou 50 %

Notez que `\np` est un alias rendu disponible en passant l'option `np` lors de l'inclusion du package `numprint`.

Ces packages sont assez complets, et je vous renvoie à leur documentation pour les détails. On peut notamment changer quasiment tous les séparateurs, signes, espaces, etc. Personnellement, par exemple, j'essaye de suivre les recommandations de l'imprimerie nationale sur les séparateurs de milliers et demande donc à `numprint` d'utiliser dans ce rôle des espaces fines insécables (`\,`) alors que sa valeur par défaut pour le français est `~` :

Avant : 1 234 567

`\npthousandsep{\,}`

Après : 1 234 567

L'un des principaux intérêts de ces packages réside en les outils qu'ils fournissent pour présenter des nombres dans des tableaux. Pour cet usage précis, je commence à préférer `siunitx`, qui gère un peu mieux l'alignement horizontal, la mise en forme et l'introduction de « contenu parasite » dans les cellules.

```
\begin{tabular}{
            S[table-format = 5.2]
            n{5}{2}
        }
    \toprule
    \multicolumn{1}{c}{Siunitx} &
        \multicolumn{1}{c}{Numprint} \\
    \midrule
    12345.67 & 12345.67 \\
      654.3  &   654.3  \\
    \color{red} 12 & {\color{red}} 12 \\
    \bottomrule
\end{tabular}
```

Siunitx	Numprint
12 345,67	12 345,67
654,3	654,3
12	12

Ces packages définissent des types de colonnes qui permettent par exemple d'aligner les nombres sur la virgule, de faire automatiquement des arrondis, etc. Là encore, allez voir la documentation. Notez aussi que pour `numprint` certaines commandes de mise en forme (voir le `\color` de l'exemple) doivent être protégées par des accolades pour éviter qu'elles ne soient interprétées comme une partie des données à afficher, tandis que `siunitx` inclut des mécanismes pour reconnaître certaines de ces commandes et s'en sort donc bien souvent mieux sans ces accolades qu'avec.

Si vous vous aventurez dans le délire de « demander à ces packages de mettre certaines valeurs en évidence avec du gras tout en conservant l'alignement des nombres », il faut juste être conscient que certaines polices d'écriture ne possèdent pas de version « mode maths gras spécial où tous les chiffres occupent la même largeur ». J'ai parfois un peu lutté avec Beamer qui fait des remplacements de polices à sa sauce.