Références

[1] Barbara Beeton. « What every (LA)TEX newbie should know ». In : *TUGboat* 44.2 (2023), p. 164-169. DOI : 10.47397/tb/44-2/tb137beeton-basic. URL : https://doi.org/10.47397/tb/44-2/tb137beeton-basic.

[2] Barbara Beeton. « Debugging LATEX files — Illegitimi non corborundum ». In : *TUGboat* 38.2 (2017), p. 159-164. URL : https://tug.org/TUGboat/tb38-2/tb119beet.pdf.

Barbara Beeton

BRÈVE INTRODUCTION À UNE COMPILATION ASSISTÉE, GRÂCE À ARARA

Introduction

Présentation rapide

Le logiciel arara est un utilitaire, intégré aux distributions LaTeX, qui permet d'automatiser des actions (définies par des règles) de compilation de fichier TeX.

Il existe d'autres assistants similaires, comme latexmk, rubber, ou encore spix. Chacun a ses spécificités propres, ses domaines d'applications, ses avantages et ses inconvénients, et nous ne rentrerons pas dans la comparaison détaillée de chacun.

Une spécificité de arara est d'utiliser `Java` comme moteur d'exécution, donc une machine `Java` est nécessaire au bon fonctionnement de cette méthode.

Pourquoi ?

Ayant parfois (souvent ?) besoin d'utiliser des chaînes de compilations différentes (qui utilisent l'option `shell-escape`, ou les moteurs `pdflatex`, `lualatex`, etc.), j'avais paramétré différentes chaînes de compilation dans mon éditeur `TeXstudio`, mais je n'étais que *moyennement* satisfait de cette méthode !

Donc une petite recherche sur des *outils* de compilation m'a conduit à me pencher sur des solutions de type « *assistants de compilation* ».

Le logiciel arara a retenu mon attention par :

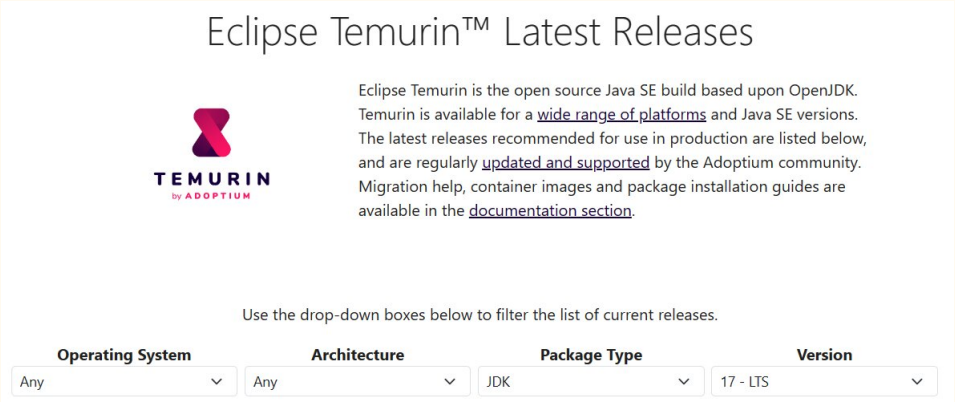
- sa simplicité de configuration, directement dans le document TeX ;
- sa simplicité à s'adapter à mes différents besoins.

Installation(s)

Pour ce qui est du *package* arara, rien de plus simple. Normalement, il est fourni par les distributions LaTeX classiques et il suffit alors de vérifier dans le gestionnaire de paquets de sa distribution LaTeX qu'il est installé, sinon l'installer grâce à ce même gestionnaire (ligne de commandes ou interface graphique).

Pour ce qui est de l'environnement `Java`, je conseille l'utilisation de la solution `Adoptium` (multi-plateformes) qui installera une machine virtuelle, et avec une inscription des exécutables dans le `PATH` c'est encore mieux !

`https://adoptium.net/temurin/releases/`



Vérification(s)

On peut vérifier que tout est prêt pour utiliser `Java` : il suffit de lancer un petit `java --version` dans un terminal qui permettra de vérifier que tout est prêt de ce côté.

Sous Ubuntu, cela donne :

```
$ java --version
%
openjdk 17.0.8 2023-07-18
OpenJDK Runtime Environment (build 17.0.8+7-Ubuntu-123.04)
OpenJDK 64-Bit Server VM (build 17.0.8+7-Ubuntu-123.04, mixed mode,
    sharing)
```

Sous Windows, cela donne :

```
$ java --version
%
Microsoft Windows [version 10.0.22621.2134]
(c) Microsoft Corporation. Tous droits réservés.
openjdk 17.0.3 2022-04-19
OpenJDK Runtime Environment Temurin-17.0.3+7 (build 17.0.3+7)
OpenJDK 64-Bit Server VM Temurin-17.0.3+7 (build 17.0.3+7, mixed
    mode, sharing)
```

On peut maintenant vérifier que tout est prêt pour utiliser arara en lançant `arara --version` dans un terminal qui permet de vérifier que tout est prêt de ce côté également.

```
$ arara --version
%
```

```
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

arara 7.1.0
Copyright (c) 2023, Island of TeX
arara is released under the New BSD license.

New features in version 7.1.0:
* Add (Lua) project support to arara.
* Use a defined domain-specific file system API instead of `java.io.
    File`.
* Use header mode by default (`-w` restores the old behavior).

See the full changelog of this release at
https://gitlab.com/islandoftex/arara/-/blob/development/CHANGELOG.md
```

Configuration et utilisation

Idée générale

L'idée générale de arara est de lire les informations de compilation dans le fichier TEX grâce à des commandes, sous forme de commentaires, placées au début du fichier, *commandes* qui seront déclarées sous la forme suivante :

Exemple 28

```
% arara: ⟨commande 1⟩
% arara: ⟨commande 2⟩
...

\documentclass{...}
...
```

Il suffira ensuite de compiler le fichier TEX à l'aide d'une *simple* ligne de compilation :

```
$ arara fichier.tex
```

Si on souhaite avoir plus d'informations lors de la compilation, on pourra utiliser l'option `-v` :

```
$ arara -v fichier.tex
```

Le logiciel arara est très *puissant* et paramétrable ! L'idée est ici de montrer des utilisations basiques, mais il est à noter que arara peut être considéré comme un outil de *scripting* (très) évolué.

La documentation permet de se rendre compte de la richesse de cet outil.

Première utilisation : travailler sur le moteur de compilation

On peut déjà utiliser arara en précisant le moteur de compilation (et ses éventuelles options), ça sera déjà une bonne base.

Dans ce cas la règle se présentera sous la forme suivante :

Exemple 29

```
% arara: ⟨moteur⟩

\documentclass{...}
...
```

Ainsi, pour compiler un document (très simple) avec `pdflatex`, il suffira de :

- mettre `% arara: pdflatex` au début du fichier ;
- compiler le fichier grâce à `arara fichier.tex`.

Par exemple, la compilation arara du document `testarara.tex` suivant :

Exemple 30

```
% arara: pdflatex
\documentclass{article}

\begin{document}

Hello world !

\end{document}
```

donnera :

```
$ arara testarara.tex
%
__ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "testarara.tex" (size: 97 B, last modified: 2023-08-11
    19:57:13), please wait.

(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS

Total: 1.44 seconds
```

De la même façon, la compilation arara du document `testarara.tex` suivant :

Exemple 31

```
% arara: lualatex
\documentclass{article}

\begin{document}

Hello world !

\end{document}
```

produira la sortie suivante :

```
$ arara -v testarara.tex
%

__ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "testarara.tex" (size: 97 B, last modified: 2023-08-11
20:02:57), please wait.

-----------------------------------------------------------------
(LuaLaTeX) LuaLaTeX engine
-----------------------------------------------------------------
This is LuaHBTeX, Version 1.17.0 (TeX Live 2023)
restricted system commands enabled.
(./testarara.tex
LaTeX2e <2023-06-01> patch level 1
L3 programming layer <2023-08-03>
(c:/texlive/2023/texmf-dist/tex/latex/base/article.cls
Document Class: article 2023/05/17 v1.4n Standard LaTeX document
    class
(c:/texlive/2023/texmf-dist/tex/latex/base/size10.clo))
(c:/texlive/2023/texmf-dist/tex/latex/l3backend/l3backend-luatex.def)

(./testarara.aux) (c:/texlive/2023/texmf-dist/tex/latex/base/ts1cmr.
    fd)
[1{c:/texlive/2023/texmf-var/fonts/map/pdftex/updmap/pdftex.map}]
(./testarara.aux))
406 words of node memory still in use:
3 hlist, 1 vlist, 1 rule, 2 glue, 3 kern, 1 glyph, 4 attribute, 48
    glue_spec
, 4 attribute_list, 1 write nodes
avail lists: 2:22,3:4,4:2,5:23,6:2,7:54,9:18
<c:/texlive/2023/texmf-dist/fonts/opentype/public/lm/lmroman10-
    regular.otf>
Output written on testarara.pdf (1 page, 3356 bytes).
Transcript written on testarara.log.

------------------------------------------------------------ SUCCESS

Total: 2.94 seconds
```

Deuxième utilisation : travailler sur les options du moteur

On peut également spécifier des options au compilateur, grâce à la syntaxe suivante :

Exemple 32

```
% arara: ⟨moteur⟩: {⟨option 1⟩: ⟨val. 1⟩, ⟨option 2⟩: ⟨val. 2⟩, ...}

\documentclass{...}
...
```

Ainsi on va pouvoir – directement dans le fichier TEX – paramétrer ces options. Fini les lignes de commandes interminables pour compiler les documents. De plus, il arrive très régulièrement, pour se souvenir de la chaîne de production d'un document, de s'écrire un *mémo* en commentaire au début du document. Ici avec arara, cela fait double emploi : on note la chaîne de production de telle façon qu'arara puisse l'interpréter et automatise le travail.

Ainsi, la règle suivante :

Exemple 33

```
% arara: pdflatex: {shell: yes, synctex: no, interaction: batchmode}
```

demandera de compiler en `pdflatex` :

- avec un accès `shell-escape` ;
- sans la création du fichier `synctex` ;
- avec une interaction de type `batchmode`.

Avec cette commande arara,la compilation arara suivante :

```
$ arara testarara.tex
```

est équivalente à la ligne de commande suivante :

```
$ pdflatex --shell-escape --synctex=0 --interaction=batchmode
    testarara.tex
```

Un peu de compilation « conditionnelle »

Une des forces de arara est de pouvoir créer et utiliser les *règles* incluant des tests (la documentation permet de se rendre compte des multiples possibilités !).

Pour ma part, je me sers de arara pour compiler plusieurs fois si nécessaire (références croisées, sommaire, etc.) [45], et dans ce cas je paramètre arara comme suit :

Exemple 34

```
% arara: pdflatex
% arara: pdflatex if found('log', '(undefined references|Please
    rerun|Rerun to get)')

\documentclass{article}

\begin{document}

Hello world !

\end{document}
```

Autrement dit :

- la première règle va compiler (sans condition) avec `pdflatex` ;
- la deuxième règle va compiler avec `pdflatex` uniquement si `undefined references`, `Please rerun` ou `Rerun to get` est trouvé dans le fichier `log`.

En compilant avec arara, on obtient :

45. Ici, latexmk est sans aucun doute bien plus puissant, cependant, il fonctionne en boîte noire.

```
$ arara testarara.tex
%

__ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "testarara.tex" (size: 186 B, last modified:
2023-08-13 13:12:11), please wait.

(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS
(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS

Total: 2.982 seconds
```

Si on compile une deuxième fois, on se rend compte que la deuxième règle n'est pas exécutée :

```
$ arara testarara.tex
%

  __ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "testarara.tex" (size: 186 B, last modified:
2023-08-13 13:12:11), please wait.

(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS

Total: 1.782 seconds
```

Dans le cas de l'utilisation d'une bibliographie, on pourra utiliser les règles suivantes (à compléter si besoin) :

Exemple 35

```
% arara: pdflatex
% arara: bibtex
% arara: pdflatex
% arara: pdflatex

\documentclass{article}

\begin{document}
...
...
```

En compilant avec arara, la sortie donnera :

```
$ arara document.tex
%

__ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
```

```
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "document.tex" (size: 186 B, last modified:
2023-08-13 13:12:11), please wait.

(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS
(BibTeX) The BibTeX reference management software ....... SUCCESS
(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS
(PDFLaTeX) PDFLaTeX engine ............................. SUCCESS
Total: 2.49 seconds
```

Compilation de figures externes

Grâce à arara, on peut aussi interfacer plusieurs programmes [46]. Lorsqu'on a des figures externes produites avec METAPOST, Asymptote, etc., il est possible, à partir de son source TEX, de demander à arara de compiler les figures si les sources de celles-ci ont été modifiées.

Ici, nous montrons un exemple avec METAPOST. Considérons le code source suivant :

Exemple 36

```
% arara: metapost: { files: [fig.mp] } if changed(toFile('fig.mp'))
% arara: latexmk: { engine: lualatex }

\documentclass{article}
\usepackage{graphicx}
\DeclareGraphicsRule{*}{mps}{*}{}

\begin{document}

\includegraphics{fig.0}

\end{document}
```

La première règle indique qui faut compiler le fichier externe `fig.mp` avec METAPOST si celui-ci a été modifié depuis la dernière compilation, la deuxième indique qu'il faut compiler le fichier TEX *courant* avec latexmk en utilisant le moteur lualatex.

Ansi, la première compilation donne :

```
$ arara test_metapost.tex
%
__ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "test_metapost.tex" (size: 247 B, last modified:
2023-09-15 14:17:37), please wait.
```

46. Contrairement à latexmk par exemple.

```
(Metapost) Metapost ..................................... SUCCESS
(LaTeXmk) Tool LaTeXmk .................................. SUCCESS

Total: 1.515 seconds
```

Puis une seconde, sur le même fichier donne :

```
$ arara test_metapost.tex
%
__ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "test_metapost.tex" (size: 247 B, last modified:
2023-09-15 14:17:37), please wait.

(LaTeXmk) Tool LaTeXmk .................................. SUCCESS

Total: 0.450 seconds
```

Ici, latexmk est quand même lancé, même s'il détecte qu'il n'y pas pas de compilation à faire avec `lualatex`.

Compléments

Erreurs et fichiers log

Lors de la compilation avec arara en ligne de commandes (en mode silencieux), la console n'affichera pas forcément les erreurs de manière classique (en fonction du paramétrage de `interaction` notamment).

Donc attention – en ligne de commandes – à la gestion des erreurs : le fichier `log` est – quoi qu'il arrive – une bonne source pour analyser des erreurs.

Là encore, l'option `-v` aide à obtenir plus d'informations, comme le montrent les deux sorties suivantes.

```
$ arara testarara.tex
%
  __ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "testarara.tex" (size: 109 B, last modified:
2023-08-14 08:09:30), please wait.

(LuaLaTeX) LuaLaTeX engine .............................. FAILURE
Total: 3.254 seconds
C:\texlive\2023\bin\windows\runscript.tlu:921: command failed with
    exit code 1:
java.exe -jar c:\texlive\2023\texmf-dist\scripts\arara\arara.jar
    testarara.tex
```

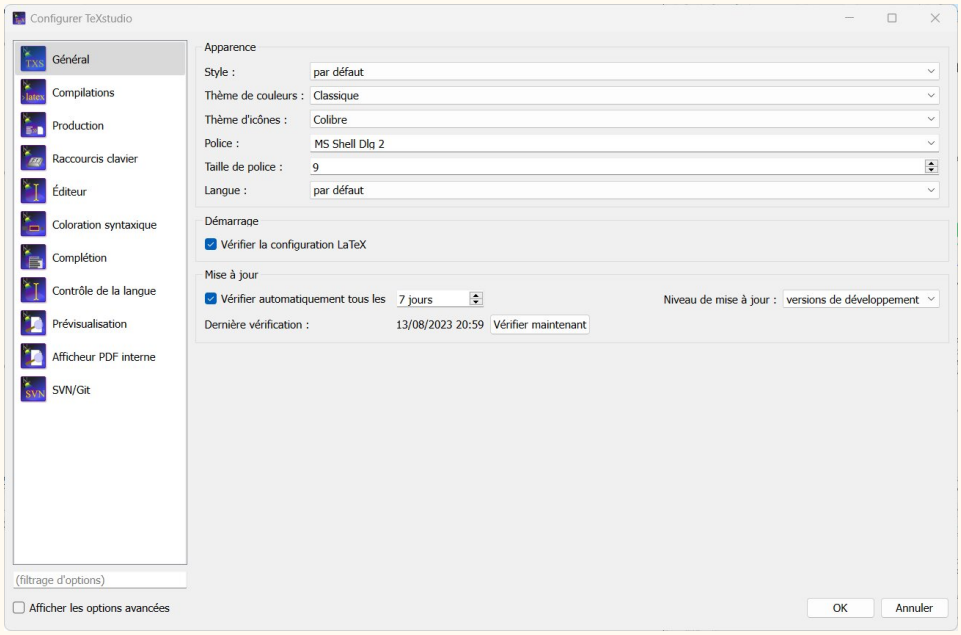
```
$ arara -v testarara.tex
%
  __ _ _ __ __ _ _ __ __ _
/ _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
\__,_|_|  \__,_|_|  \__,_|

Processing "testarara.tex" (size: 109 B, last modified:
2023-08-14 08:09:30), please wait.

----------------------------------------------------------------
(LuaLaTeX) LuaLaTeX engine
----------------------------------------------------------------
This is LuaHBTeX, Version 1.17.0 (TeX Live 2023)
restricted system commands enabled.
(./testarara.tex
LaTeX2e <2023-06-01> patch level 1
L3 programming layer <2023-08-11>
(c:/texlive/2023/texmf-dist/tex/latex/base/article.cls
Document Class: article 2023/05/17 v1.4n Standard LaTeX document
    class
(c:/texlive/2023/texmf-dist/tex/latex/base/size10.clo))
(c:/texlive/2023/texmf-dist/tex/latex/l3backend/l3backend-luatex.def)

(./testarara.aux) (c:/texlive/2023/texmf-dist/tex/latex/base/ts1cmr.
    fd))
Runaway argument?
{$ \par \end {document}
! File ended while scanning use of \frac .
<inserted text>
\par
<*> testarara.tex

?
}
```

Interaction avec un éditeur de fichiers comme `TeXstudio`

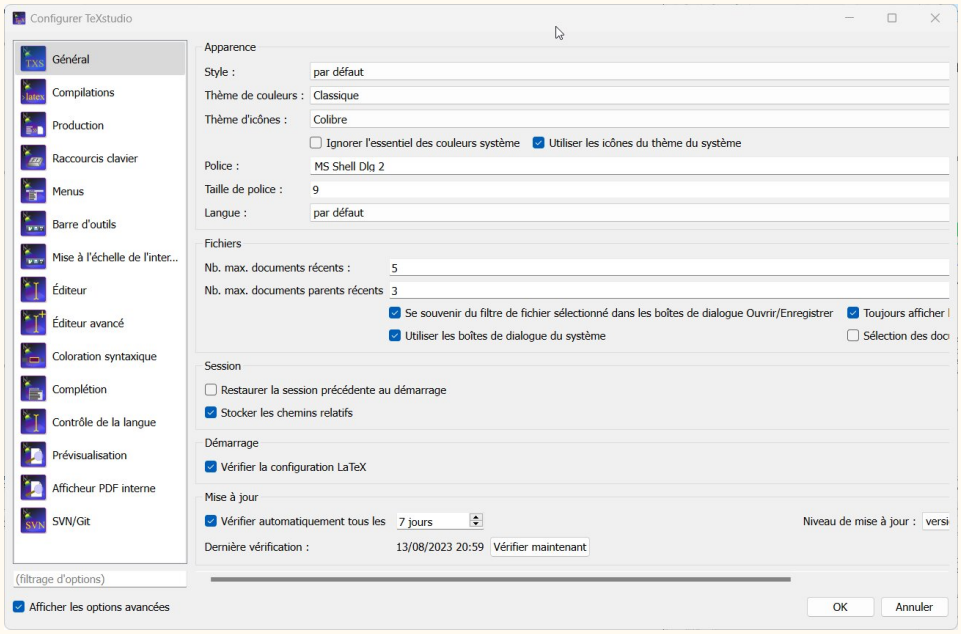
L'éditeur `TeXstudio` permet de travailler également avec arara, en le déclarant comme compilateur par défaut (et en utilisant éventuellement un *commentaire magique*).

Pour cela, il y a quelques petites manipulations à faire :

– on va dans le menu : Options » Configuration TeXstudio ;

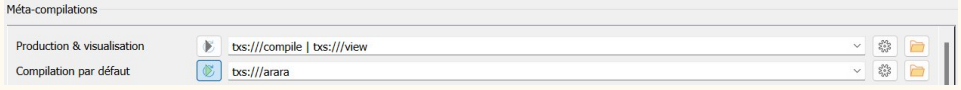


– on coche (si besoin) la case Afficher les options avancées :



– dans l'onglet Production, on crée une nouvelle compilation utilisateur :

– on peut maintenant configurer Compilation par défaut :



Il suffit ensuite de compiler le fichier TEX grâce à F5 (Compilation & Production) !