Arara — TEX automation made easy

Paulo Roberto Massa Cereda

Abstract

This article covers a bit of history behind arara, the cool TEX automation tool, from the earlier stages of development to the new 4.0 series. We also highlight some noteworthy features of our tool.

1 Introduction

Writing software is easy. Writing *good* software is extremely difficult. I was working on a Catholic songbook with 1200+ songs and several indices and cross-references. The compilation steps required to achieve the final result were getting out of hand.

At some point, I realized I *knew* all the steps I had to reproduce beforehand, I only had to find a way to automate them! Inspired by the way compilers work (i.e., read a source file, ignore all comments and process the rest), I could exploit TEX comments to include special indications on what to do on the document. Since engines do ignore comments, no side effects would arise, at least document-wise.

It was a cold afternoon. I sat in front of my computer and decided to work on this new tool. It was a matter of time to reach preliminary yet promising results. I mentioned this effort in the chat room of the TEX community at StackExchange and some friends asked me to make a public release out of it, as other users could benefit from this new tool.

However, a name was needed for the tool. In the chat room, we used to have a lot of fun with palindromes (especially palindromic reputations in arbitrary bases), so I took that aspect as inspiration. Then I thought of a very beautiful, colourful bird of the Brazilian fauna: the macaw, or as we like to call it, the arara. The name was immediately adopted!

Once the name was chosen, I needed a logo. Since I am a Fedora Linux user, I was always a fan of their default typeface, which is quite round! The choice was made: the humble arara tool became arara! (But we'll use the more subdued arara in regular text.) My life was about to change.

2 A bit of history

A lot of things have happened since version 1.0, released in 2012, to the new version 4.0, released in 2018. This section presents a bit of history of arara, including challenges in each version.

2.1 The first version

There is a famous quote along the lines of "If at first you do not succeed, call it version 1.0." The first version of arara was also the first public release, dated April 2012. Nothing much was there, besides the core concepts that still exist today:

- *Rules:* a rule is a formal description of how arara handles a certain task. It tells the tool how to do something.
- *Directives:* A directive is a special comment inserted in the source file in which you indicate how arara should behave.

Back then, we could write directives in our document and have the tool process them as expected, like the following example:

```
% arara: pdftex
Hello world!
\bye
```

Amusingly, the first version offered only a log output as an additional feature. There was no verbose mode. The log file was a gathering of streams (error and output) from the sequence of commands specified through directives. And that was it.

2.2 The second version

The first version had a serious drawback: compilation feedback was not in real time and, consequently, no user input was allowed. For the second version, real time feedback was introduced when the tool was executed in verbose mode.

```
$ arara -v mydoc.tex
... [real time feedback] ...
```

Two other features were included in this version: a flag to set an overall execution timeout, in milliseconds, as a means to prevent a potentially infinite execution, and a special variable in the rule context for handling cross-platform operations.

2.3 The third version

So far, arara was only a tiny project with a very restricted user base. However, for version 3.0, a qualitative goal was reached: the tool became international, with localised messages in English, Brazilian Portuguese, German, Italian, Spanish, French, Turkish and Russian. Further, new features such as configuration file support and rule methods brought arara to new heights. As a direct consequence, the lines of code virtually doubled from previous releases.

```
$ arara --help -L es
...
 -h,--help     imprime el mensaje de ayuda
 -l,--log      genera el registro de la salida
...
```

When the counter stopped at version 3.0, Brent Longborough, Marco Daniel and I decided it was time for arara to graduate and finally be released in TEX Live. Then things really changed in my life. The

tool was a success! Given the worldwide coverage of that TEX distribution, arara silently became part of the daily typographic tool belt of many users. But then, the inevitable happened: a lot of bugs emerged from the dark depths of my humble code.

2.4 Critical and blocker bugs

Suddenly, several questions about arara were posted in the TEX community at StackExchange and I was not able to provide a consistent, definitive answer for many of them! It was very tricky to track the bugs to their sources, and some of them were really nasty. For instance, a simple scenario of a file with spaces in the name was more than enough to make the poor tool cry for help for apparently no reason:

```
$ arara "My PhD thesis.tex"
```

Likewise, the issues page of the project repository hosted at GitHub had a plethora of reports, and little could I do about them. I delved into the code of third party libraries, but the root of all evil seemed to lie in my own sources.

2.5 Nightingale

In all seriousness, I was about to give up. My code was not awful, but there were a couple of critical and blocking bugs. Something very drastic had to be done in order to put arara back on track. Then, proceeding on faith, I decided to rewrite the tool entirely from scratch. In order to achieve this goal, I created a sandbox and started working on the new code. And this new project got a proper name: nightingale.

It was the right thing to do. Nicola Talbot helped me with the new version, writing code, fixing bugs and suggesting new features. She was writing a book about LATEX for administrative work at the time and was extensively using arara in the code examples. Her writing indirectly became my writing as well, as I progressively improved the code and added new features to match her suggestions.

2.6 The fourth version

At some point, nightingale had to say farewell and gave most of its features to the bigger, older bird in the nest. It is worth mentioning that nightingale still lives in my repository at GitHub for those who are bold enough to try it. From 1500+ lines of code in version 3.0, arara 4.0 tripled that number: a whopping 4500+ lines of code! And, most important: all critical and blocking bugs were completely fixed.

However, although the code was ready for production, the user manual was far from being finished. In fact, the documentation had to be written entirely from scratch. Then another saga started: find proper time and effort to document a great yet complex tool in all details, from user to developer perspectives.

It took me a lot of dedication to write the user manual and try to cover as much detail as possible for every feature, old and new, and the tool itself. Some of the internals had to be changed, so more explanations were needed. Documenting a tool is almost as difficult as writing code for it!

3 New features

This section highlights some noteworthy features found in the new version 4.0 of arara. For additional information, please refer to our user manual.

3.1 REPL work flow

In version 4.0, arara employs a REPL (read-evaluate-print loop) work flow for rules and directives. In previous versions, directives were extracted, their corresponding rules were analyzed, commands were built and added to a queue before any proper execution or evaluation. I decided to change this work flow, so now arara evaluates each rule on demand, i.e., there is no *a priori* checking. A rule will always reflect the current state, including potential side effects from previously executed rules.

3.2 Multiline directives

Sometimes, directives can span several source lines, particularly those with several parameters. From arara 4.0 on, we can split a directive into multiple lines by using the `arara: -->` mark on each line which should comprise the directive. We call it a multiline directive. Let us see an example:

```
% arara: pdflatex: {
% arara: --> shell: yes,
% arara: --> synctex: yes
% arara: --> }
```

It is important to observe that there is no need for them to be on contiguous lines in the source file, i.e., provided that the syntax for parameterized directives holds for the line composition, lines can be distributed all over the code. The log file (when enabled) will contain a list of all line numbers that made up a directive.

3.3 Directive conditionals

arara 4.0 provides logical expressions, written in the MVEL language, and special operators processed at runtime in order to determine whether and how a directive should be processed. This feature is named *directive conditional*, or simply *conditional* for short. The following list describes all conditional operators available in the directive context.

- `if`: The associated MVEL expression is evaluated beforehand, and the directive is interpreted if, and only if, the result of such evaluation is true. This directive, when the conditional holds true, is executed at most once.

```
% arara: pdflatex if missing('pdf')
% arara: --> || changed('tex')
```

- `unless`: Same as `if` but the condition test is inverted.

```
% arara: pdflatex unless unchanged('tex')
% arara: --> && exists('pdf')
```

- `until`: The directive is interpreted the first time, then the associated MVEL expression evaluation is done. As long as the result holds false, the directive is reinterpreted. There is no guarantee of halting.

```
% arara: pdflatex until !found('log',
% arara: --> 'undefined references')
```

- `while`: Same as `until` but the condition test is inverted.

```
% arara: pdflatex while missing('pdf')
% arara: --> || found('log', 'undefined
% arara: --> references')
```

Although there is no conceptual guarantee for proper halting of unbounded loops, we have provided a practical solution to potentially infinite iterations: arara has a predefined maximum number of loops. The default value is 10, but it can be overridden either in the configuration file or on the command line.

3.4 Directive extraction only in the header

The `--header` command line option changes the mechanics of how arara extracts the directives from the code. The tool always reads the entire file and extracts every single directive found throughout the code. However, by activating this switch, arara will extract all directives from the beginning of the file until it reaches a line that is not empty and is not a comment (hence the option name). Consider the following example:

```
% arara: pdftex
Hello world.
\bye
% arara: pdftex
```

When running arara without the `--header` option, two directives will be extracted (on lines 1 and 4). However, if executed with this switch, the tool will only extract one directive (from line 1), as it will stop the extraction process as soon as it reaches line 2.

3.5 Dry-run execution

The `--dry-run` command line option makes arara go through all the motions of running tasks and subtasks, but with no actual calls. This is useful for testing the sequence of underlying system commands to be performed on a file.

```
[DR] (PDFLaTeX) PDFLaTeX engine
---------------------------------------
Authors: Marco Daniel, Paulo Cereda
About to run: [ pdflatex, hello.tex ]
```

Note that by the rule, authors are displayed (so they can be blamed in case anything goes wrong), as well as the system command to be executed. It is an interesting approach to see everything that will happen to your document and in which order. It is important to observe, though, that conditionals are not evaluated in this mode.

3.6 Local configuration files

From version 4.0 on, arara provides support for local configuration files. In this approach, a configuration file can be located in the working directory associated with the current execution. This directory can also be interpreted as the one relative to the processed file. This approach offers a project-based solution for complex work flows, e.g., a thesis or a book. However, arara must be executed within the working directory, or the local configuration file lookup will fail. Observe that this approach has the highest lookup priority, which means that it will always supersede a global configuration.

3.7 File hashing

arara 4.0 features four methods for file hashing in the rule and directive scopes, presented as follows. The file base name refers to the file name without the associated extension.

- `changed(extension)`: checks if the file base name concatenated with the provided extension has changed its checksum from last verification.
- `changed(file)`: the very same idea as the previous method, but with a proper Java `File` object instead.
- `unchanged(extension)`: checks if the file base name concatenated with the provided extension is unchanged from last verification. It is the opposite of the `changed(...)` method.
- `unchanged(file)`: the very same idea as the previous method, but with a proper Java `File` object instead.

The value is stored in a database file named `arara.xml` as a pair containing the full path of the provided file and its corresponding CRC-32 hash (the

database is created as needed). If the entry already exists, the value is updated, or created otherwise.

3.8 Dialog boxes

A *dialog box* is a graphical control element, typically a small window, that communicates information to the user and prompts them for a response. arara 4.0 provides UI methods related to such interactions. As good practice, make sure to provide descriptive messages to be placed in dialog boxes in order to ease and enhance the user experience.

3.9 Session

Rules are designed under the *encapsulation* notion, such that direct access to the internal workings of such structures is restricted. However, as a means of supporting framework awareness, arara provides a mechanism for data sharing across rule contexts, implemented as a `Session` object. In practical terms, this particular object is a global, persistent map composed of keys and values available throughout the entire execution.

3.10 Redesigned user interface

For arara 4.0, we redesigned the interface in order to look more pleasant to the eye; after all, we work with TEX and friends. Please note that the output here is truncated to respect the column width.

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

Processing 'doc.tex' (size: 307 bytes, last
modified: 05/29/2018 08:57:30), please wait.

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

Total: 1.45 seconds
```

First of all, we have the nice application logo, displayed using ASCII art. The entire layout is based on monospaced font spacing, usually used in terminal prompts. Hopefully you follow the conventional use of a monospaced font in your terminal, otherwise the visual effect will not be so pleasant. First and foremost, arara displays details about the file being processed, including size and modification status:

```
Processing 'doc.tex' (size: 307 bytes, last
modified: 05/29/2018 08:57:30), please wait.
```

The list of tasks was also redesigned to be fully justified, and each entry displays both task and subtask names (the former being displayed enclosed in parentheses), besides the usual execution result:

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

If a task fails, arara will halt the entire execution at once and immediately report back to the user. This is an example of what a failed task looks like:

```
(PDFLaTeX) PDFLaTeX engine ..... FAILURE
```

Also, observe that our tool displays the execution time before terminating, in seconds. The execution time has a very simple precision, as it is meant to be easily readable, and should not be considered for command profiling.

```
Total: 1.45 seconds
```

The tool has two execution modes: *silent*, which is the default, and *verbose*, which prints as much information about tasks as possible:

- When in silent mode, arara will simply display the task and subtask names, as well as the execution result. Nothing more is added to the output.
- When executed in verbose mode, arara will display the underlying system command output as well, when applied. In version 4.0 of our tool, this mode was also entirely redesigned in order to avoid unnecessary clutter, so it would be easier to spot each task.

It is important to observe that, in verbose mode, arara can offer proper interaction if the system command requires user intervention. However, in silent mode the tool will simply discard this requirement and the command will almost surely fail.

4 The future

Now that arara 4.0 is officially released and already available in CTAN and TEX Live, it is time to plan the future. Our repository already has suggestions for new features and improvements. The work on arara 5.0 has begun! If you have any feedback about our tool, please drop us a note.

Also, if you believe your custom rule is comprehensive enough and deserves to be in the official pack, please contact us. We will be more than happy to discuss the inclusion of your rule in forthcoming updates. Happy TEXing with arara!

◇ Paulo Roberto Massa Cereda
Analândia, São Paulo, Brazil
cereda dot paulo (at) gmail dot com
github.com/cereda/arara