
Web page to PDF conversion with Rmodepdf: Leveraging Lua \LaTeX for e-book reader-friendly documents

Michal Hoftich

Abstract

This article presents the use of responsive design methods and advanced features of Lua \LaTeX for automatic document typesetting intended for various target outputs, both printed and electronic, such as mobile phones, tablets, or e-readers.

Specifically, it focuses on the use of Lua \LaTeX for automated typesetting with the help of the Responsive package [7] for setting font size and line spacing according to page size, the Luavlna package [5] to prevent single-character prepositions at the ends of lines, the Lua-widow-control package [2] to minimize widows and orphans at the ends and beginnings of pages, and the Linebreaker package [4] to prevent line overflow.

1 Introduction

Some time ago, I acquired an e-book reader, but I still read most texts on my PC screen because they come from web sources. It occurred to me that I could save longer articles for later reading on my e-reader. There are, of course, several applications for this purpose, but I decided to create my own, tailored exactly to my needs and preferences. Another motivation is the opportunity to learn something new and create packages that could be useful for other \TeX users as well.

My goal is to make the solution as automated as possible, so I don't have to deal with overfull lines or other errors that would require manual intervention. Thanks to the capabilities of Lua \LaTeX , such a solution is possible today, as we will demonstrate in the following text.

Because Lua \TeX provides the Lua programming language in \TeX distributions, I used it to create my project Rmodepdf [8]. It uses the LuaXML package [6] to transform HTML into \TeX and a few external commands — Curl for downloading of the web pages, and Rdrview [3], which removes navigation elements, advertisements, and other distractions from the page. Rdrview is based on the JavaScript library used by the Firefox browser for its Reader Mode. However, it has been translated from JavaScript to C, making it significantly faster and eliminating the need for any additional dependencies.

During the development of the program, I also created or significantly expanded three \LaTeX packages that may be useful on their own. For the LuaXML package, I created an HTML parser that allows

web pages to be processed directly from the Lua language. The Responsive package enables the creation of templates that adjust font size, page margins, and other parameters according to the current page size. Finally, the Linebreaker package prevents line overflow, which is crucial in automated document typesetting where we neither want nor can manually correct such errors.

In addition to saving articles for reading on an e-reader, there are other ways to utilize the Rmodepdf program. One such use is archiving web content on paper. By removing all navigation elements, we obtain a document that can be easily printed, bound, and archived as a book.

2 Basic usage of Rmodepdf

The `rmodepdf` command accepts one or more urls as arguments. It is also possible to use the addresses of local HTML files.

```
$ rmodepdf <url1> <url2>
```

The output file is named based on the title of the first document. If the title cannot be found, a name based on the current date and time is chosen. The generated name is displayed in the terminal output. You can specify a custom file name using the `-o` option.

If you prefer not to compile the document directly but only to display the text generated by the \LaTeX document, you can use the `-p` option.

```
$ rmodepdf -p <url> > foo.tex
```

You can choose a different page size using the `-P` option. By default, the page size and margins are set for e-book readers, but you can also select other sizes, such as A4 paper size. The page style is currently set to empty (*blank*), but you can change it using the `-s` option.

```
$ rmodepdf -P a4paper -s plain <url>
```

For speed, images are stored in a local directory. By default, this is the `img/` subdirectory within the current directory, but you can specify a different directory using the `-i` option.

```
# save the document as foo.pdf and
# save images in the temp dir
$ rmodepdf -o foo.pdf -i /tmp/img <url>
```

You can disable image downloading entirely with the `-n` option. Rmodepdf also detects and displays \LaTeX mathematical commands embedded in web pages that use MathJax or KaTeX for rendering. This default behavior can be disabled using the `-N` option. Additionally, the removal of page elements using Rdrview can be disabled with the `-R` option.

3 Configuration

3.1 Settings

It often happens that during conversion you encounter errors or wish to change how certain elements on the page are converted into \LaTeX . Therefore, `Rmodepdf` provides the option to use a Lua configuration script. This script allows you to modify the code of translated pages, define conversion rules, set variables, or change templates as needed. The configuration file is loaded using the option `-c`.

```
$ rmodepdf -c script.lua <url>
```

The script might look like this, for example:

```
add_to_config {
  document = {
    preamble_extras = [[
      \setmainfont{Linux Libertine O}
    ]],
  },
  img_convert = {
    -- modify the command used for
    -- conversion of SVG images to PDF
    svg = "cairosvg -o ${dest} -",
  },
}
```

Above, the command `add_to_config` is used, which safely copies new configuration values into the original configuration. If you only want to set a single configuration value, you can also directly write to the `config` table:

```
config.document.geometry = "a6paper"
```

The `config` table contains several subtables that you can configure. The `document` subtable includes properties of the output document, such as `preamble_extras` for adding additional code to the document preamble, or `geometry`, which allows you to directly specify the dimensions of the page or margins of the output document.

The subtable `img_convert` defines commands for converting image formats used on the converted web pages that are not supported in $\text{Lua}\LaTeX$ to one of the supported formats. For example, in the sample, we define a command to convert from SVG format to PDF. This command must support reading from standard input, and you can specify the output file name using the template `${dest}`.

The subtable `html_latex` contains settings for translating \LaTeX code embedded in web pages. The `ignored` item contains a list of HTML elements where embedded \LaTeX code should not be searched for. Typically, this includes elements like `<pre>`, which contain source code that should not be processed in our document.

The subtable `pages` contains converted files and their metadata. Its content is populated after the configuration script runs, so it is not available beforehand but is utilized in templates. It includes items such as `language` for the document language, `title` for the document title, and `content` which contains the \LaTeX code of the document for transformation.

3.2 Callbacks

The configuration script is executed before the actual conversion, so it cannot directly influence the conversion process. However, we can define several callback functions that allow us to affect the conversion. These functions are as follows:

preprocess_content modify string with the raw HTML before readability and DOM parsing.

preprocess_dom modify the DOM object before fetching of images or handling of MathJax.

postprocess_dom modify the DOM after all processing by `Rmodepdf`.

postprocess late post-processing of the config table.

The most useful are the first three. The `preprocess_content` function takes a string parameter with the HTML code of the page as it was downloaded from the original website, before any modifications by `Rdrview`. Here, you can use Lua string functions to fix certain elements that may cause issues during processing with `Rdrview`. This method is quite limited and, especially when using regular expressions, it can cause more harm than good. Therefore, use it with caution.

The difference between the next two callbacks is that with the first one, you can still influence image downloading or the processing of \LaTeX commands. For modifications to the final version of the document, it is best to use `postprocess_dom`.

Both functions receive a `LuaXML` DOM object as a parameter. This allows you to safely traverse and transform the entire document. `LuaXML` includes many functions for working with the DOM; here, we will introduce just a few basics. For example, the following example prints the resulting DOM object as HTML code:

```
function postprocess_dom(dom)
  print(dom:serialize())
  return dom
end
```

The `dom:serialize()` method obtains the HTML code from the DOM object, which we then print using the `print` command. It is important to return the DOM at the end of the function; this

ensures that any modifications made to the DOM are preserved and applied to the final document.

Here’s a slightly more complex example. Let’s assume we want to remove a menu that might look like this, since Rdrview did not do the removal:

```
<div class="menu">
... menu contents ...
</div>
```

We can use the `postprocess_dom` function to remove this menu:

```
function postprocess_dom(dom)
  -- Find the menu using a CSS selector
  local menu = dom:query_selector(".menu")

  -- Iterate over the menu elements
  -- and remove each one
  for _, el in ipairs(menu) do
    el:remove_node()
  end

  -- Return the modified DOM
  return dom
end
```

In this example:

1. We use the `query_selector` method to find all elements with the class `menu`.
2. Iterate over each element retrieved in the previous step using a `for` loop.
3. Remove each menu element using the `remove_node` method.
4. Return the modified DOM at the end of the function.

This ensures that any remaining menus are removed from the final document.

3.3 Transformation from HTML to L^AT_EX

We perform the conversion of HTML elements to L^AT_EX using the `luaxml-transform` library. This library allows us to declare simple rules for transforming XML or HTML elements into text. Elements can be selected using CSS selectors, which is important because elements with the same name but different classes may need to be converted differently. For example, `` or `<div>` elements are often used as universal tags, but their intended display can vary greatly, depending on their class.

In the configuration file, the `htmlprocess` variable contains an object with rules for converting HTML elements. It provides two main functions: `htmlprocess.reset_actions`, which clears all rules for a given selector, and `htmlprocess.add_action`, which adds new rules. The following code displays some basic usage of the transformation library:

```
htmlprocess.reset_actions("br")
htmlprocess.reset_actions("figure")
htmlprocess.add_action("br", "\n\n")
htmlprocess.add_action("img",
  [[\includegraphics[max width=\textwidth]
      {@{src}}]])
htmlprocess.add_action("figure",
  "\n\n\medskip\n\n\nnoindent %s")
```

In this example, we reset the default rules for the `
` and `<figure>` elements and introduce custom rules with specific syntax. The rules adhere to the following conventions:

- The `%s` string inserts the transformed content of the element. It is crucial to include `%s` in most rules to ensure the content is correctly processed; omitting it would hide the entire element’s content. However, since the `
` element does not contain any text, it is unnecessary to use `%s` with it.
- In the rule for the `` element, `@{src}` inserts the value of the `src` attribute, which contains the image’s address. We use Lua’s double-bracket syntax for string constants to avoid C-like interpretation of the backslashes.

The following example demonstrates the use of CSS selectors for classes and attribute value comparisons to handle different types of links differently:

```
htmlprocess.reset_actions("a")
htmlprocess.add_action(
  "a.easy-footnote-to-top", "")
htmlprocess.add_action(
  'a[href="#easy-footnote"]', "%s")
```

Links with the class `a.easy-footnote-to-top` are hidden because the replacement text string is empty. However, for links whose `href` attribute starts with `#easy-footnote`, only their text content is displayed.

This was just a brief introduction to the transformation possibilities using `luaxml-transform`. You can find many more examples in the LuaXML manual.

4 Template

After converting from HTML to L^AT_EX, we need to combine the resulting code into a single document that can be compiled. Therefore, `Rmodepdf` includes a simple templating system that allows us to merge individual pages and their metadata together.

A basic template might look like this:

```
\documentclass{article}
\usepackage{linebreaker,responsive}
\usepackage[_{document.languages}%s/{,}]
{babel}
```

```

\usepackage[ $\langle$ document.geometry $\rangle$ ]{geometry}
\pagestyle[ $\langle$ document.pagestyle $\rangle$ ]
 $\langle$ document.preamble_extras $\rangle$ 
\begin{document}
   $\langle$ pages $\rangle$ 
\selectlanguage[ $\langle$ language $\rangle$ ]
?{title}{Title:  $\langle$ title $\rangle$ }\par{}}
?{author}{Author:  $\langle$ author $\rangle$ }\par{}}
\href[ $\langle$ url $\rangle$ ]{ $\langle$ url $\rangle$ }\par
 $\langle$ content $\rangle$ 
/}{\clearpage}
\end{document}

```

Templates contain three syntactic constructs. The basic one is printing a variable using \langle variablename \rangle . Variables are contained in the `config` table, and using a dot, we can also print properties of subtables. For example, \langle document.preamble_extras \rangle prints the `config.document.preamble_extras` variable.

The next construct is loops. They have the syntax \langle variablename \rangle loop code/{separator}. The variables used have to be arrays. For example, `document.languages` contains the languages of all translated documents in a format suitable for the Babel package, or `pages`, which contains all converted documents. In the loop code, variables of the currently processed array are available. If the array contains only strings, we can use the placeholder `%s`. This is used for `document.languages`. If the current object is a table, we can access its fields directly using \langle variablename \rangle .

The last construct is conditions. Their syntax is \langle variablename \rangle {true}{false}. In the example, we use them to insert the title and author, because not all pages have these items.

Custom templates can be read using the `-t` option.

```
$ rmodepdf -t mytemplate.tex  $\langle$ url $\rangle$ 
```

5 Automatic typesetting

This brings us to the next part. Since `Rmodepdf` compiles web pages directly into PDF, we cannot easily intervene in the conversion process. Therefore, the conversion needs to be as automated and error-free as possible. The output PDF can also have various page sizes. The default size is adapted for e-book readers, but we might also want to create a PDF suitable for smartphones or, conversely, a standard A4 size. For all these sizes, we need to choose different font sizes or page margins. This can be achieved using two new packages, which we will demonstrate in this section.

5.1 Responsive design

One of the issues that needs to be addressed is setting the correct font size for readability. The default font size in L^AT_EX is 10 points, regardless of the page size. This is a suitable font size for an A5 page. For A4 format, the font size should be larger, and for smaller screens of e-readers and mobile phones, it can be smaller. Similarly, we can change the line spacing, which also affects text readability depending on the font size and page size.

Web browsers face a similar problem, as they need to display text on large PC monitors as well as on the smaller screens of laptops, tablets, and mobile phones. The solution they use is called *responsive design*.

Responsive design is a way of designing web pages that allows flexible and dynamic adaptation of the appearance and layout of the page content to different display devices. One of the key elements of responsive design is a flexible structure that allows elements on the page to be resized to fit the display device.

Another important element is *media queries*. These allow defining rules that apply based on the properties of the display device, such as screen width and height or the type of output (paper, display). Thanks to these rules, the same page code can be displayed well on both large monitors and mobile devices or when printed.

The Responsive package [7] is inspired by these principles. Its main function is to set the font size according to the page size and the approximate number of characters that should fit on a line. It also sets the typographic scale [9] (affecting font sizes for headings or footnotes), the font baseline, and supports a simple version of media queries.

5.1.1 Setting up the Responsive package

Responsive automatically sets the font size, line spacing, and typographic scale at the beginning of the document. Default values can be changed using package options:

```
\usepackage[ $\langle$ options $\rangle$ ]{responsive}
```

or the `\ResponsiveSetup{ \langle options \rangle }` command. The `\ResponsiveSetup` command can also be used directly in the document text, for example, for local font settings changes.

The Responsive package offers the following options:

noautomatic prevents automatic setting of font size and line spacing at the beginning of the document.

characters number of characters for automatic font size setting.

scale typographic scale used for font sizes.

lineratio ratio used in line spacing calculation.

5.1.2 Basic font size

The font size can be set using the `\setsizes` command. Responsive tries to set the font size so that the desired number of characters fits on a line on average. The actual number of characters depends on the text used, as each letter has a different width when using proportional fonts. In practice, the number of characters displayed on a line may be slightly higher.

If the number of characters is not specified in the `\setsizes` command, the value of the `characters` option is used. The following example uses this option setting. Figure 1 shows how the same text can be displayed differently within the same frame, depending on the settings.

```
\begin{minipage}{5cm}
\ResponsiveSetup{characters=55}
\setsizes{}
\lipsum[1]
\end{minipage}
```

5.1.3 Line spacing

By default, L^AT_EX sets the line spacing to the font size multiplied by 1.2. For different fonts and page sizes, different line spacing is appropriate. Similarly, different values may be suitable for the printed and electronic versions of the document.

I was inspired by Edoardo Cavazza’s article [1] on readability and added support for setting line spacing based on the ratio of lowercase letter height and the `lineratio` variable. This ratio is obtained by the following calculation:

$$\text{line spacing} = \frac{1ex}{\text{lineratio}/100}$$

You can observe the impact of changing the `lineratio` value in Figure 2. The choice of its optimal value depends on the font used and the page size. To achieve maximum output readability, it’s advisable to compare the output using different values.

5.1.4 Media queries

Media queries are a technique that allows web developers to dynamically adapt the appearance and behavior of web pages based on various device properties, such as screen width and height, device orientation, color support, and many others. With these conditions, it is possible to create responsive and

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(a) `characters=55`

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque.

(b) `characters=25, lineratio=38`

Figure 1: Difference in font size depending on the number of characters per line

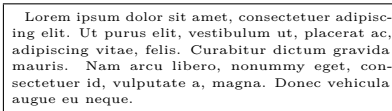
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(a) `lineratio=38`

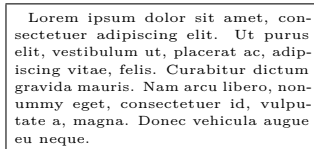
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(b) `lineratio=34`

Figure 2: Change in line spacing by changing the `lineratio` value



(a) Text width 5cm



(b) Text width 3.9cm

Figure 3: Media query example

flexible web pages that can automatically adjust to different types and sizes of devices on which they are displayed.

How can this technique be useful for L^AT_EX package authors? They could, for example, set the font size, line spacing, and other elements for specific page dimensions. After the user chooses the page size according to the device for which they want to compile the document, these elements are set automatically. The package author can define, for instance, that if the width of the text line is less than a certain size, fewer characters will be displayed on it than on longer lines. The result is shown in Figure 3.

This example will display fewer characters per line if the text width is less than 4 cm.

```
\mediaquery{max-textwidth=4cm}
  {\ResponsiveSetup{characters=45}}
  {\ResponsiveSetup{characters=60}}
```

A media query can be declared using the `\mediaquery` command, which expects three parameters: the first is a list of tests, the next parameter expects the code to be executed if the tests evaluate to true, and the last one contains the code to be executed if the condition is not met. The code can include the `\ResponsiveSetup` command, as well as any other commands. For example, setting the size of the text block, header, and footer using the geometry package.

We can test the following page properties: `paperwidth` and `paperheight` for page dimensions, `textwidth` and `textheight` for text dimensions, `orientation` for text orientation, and `twocolumn` for detecting the use of two-column text in the document.

Tests for text and page dimensions also support the prefixes `max-` and `min-`. Using these, we can test whether a given dimension is smaller or larger than a specified value.

For example, the following command changes the text color to blue if the document has landscape

The example document given below creates two pages by using Lua code alone. You will learn how to access T_EX's boxes and counters from the Lua side, shipout a page into the PDF file, create horizontal and vertical boxes (`hbox` and `vbox`), create new nodes and manipulate the nodes links structure.

(a) Without the Linebreaker package

The example document given below creates two pages by using Lua code alone. You will learn how to access T_EX's boxes and counters from the Lua side, shipout a page into the PDF file, create horizontal and vertical boxes (`hbox` and `vbox`), create new nodes and manipulate the nodes links structure.

(b) With the Linebreaker package

Figure 4: Example of using the Linebreaker package

orientation, the text width is less than 20 cm, and two columns are used.

```
\mediaquery{orientation=landscape,
  max-textwidth=20cm,
  twocolumn=true}
  {\color{blue}}
  {}
```

5.2 The Linebreaker package

The Linebreaker package [4] prevents text from overflowing in boxes and paragraphs. An example of its output is in Figure 4, where it prevents several lines from overflowing when typeset in a narrow column.

Linebreaker utilizes LuaT_EX's callback which controls line breaking. It replaces the default line breaking function with a modified version that detects overflow or underflow in the broken text. Upon detecting this problem, it retypesets the text with increased values of `\tolerance` and `\emergencystretch` until the overflow is suppressed or the maximum `\tolerance` limit is reached. These changes to `\tolerance` and `\emergencystretch` are local to the currently broken paragraph and do not affect the rest of the text.

5.2.1 Linebreaker configuration

The Linebreaker package can be configured by specifying package options using

```
\usepackage[options]{linebreaker}
```

or later in the document body with the command `\linebreakersetup{options}`. The options are:

maxcycles the number of attempts to re-typeset a paragraph.

maxemergencystretch the maximum value of `\emergencystretch`.

maxtolerance the maximum value of `\tolerance`.

For example:

```
\linebreakersetup{
  maxtolerance = 90,           % default 8189
  maxemergencystretch = 1em, % default 3em
  maxcycles = 4,              % default 30
}
```

5.3 Other packages useful for automatic typesetting

We have demonstrated the use of the Responsive and Linebreaker packages for automatic typesetting. These are not the only useful packages that leverage the power of LuaTeX for automatic typesetting. Noteworthy examples include Lua-widow-control for suppressing widows and orphans, and Luavlna, which addresses certain typographical issues for Czech and Slovak, while also preventing line breaks in SI units or academic titles.

6 Summary

I hope the demonstration of the Rmodepdf program caught your interest. Even if it didn't, I believe that the side products developed alongside it can be useful on their own.

This includes the capability to process HTML files using the LuaXML package and convert them to L^AT_EX using its `luaxml-transform` library. The Responsive package allows you to declaratively set the document design depending on the currently chosen page size. Lastly, the Linebreaker package prevents line overflow, which is a common issue especially in documents with fewer characters per line.

References

- [1] E. Cavazza. Modern CSS techniques to improve legibility. *Smashing Magazine*, 2020. www.smashingmagazine.com/2020/07/css-techniques-legibility/
- [2] M. Chernoff. *The Lua-widow-control package*. Automatically remove widows and orphans from any document. ctan.org/pkg/luawidow-control
- [3] E. Fernández. *Rdrview*. github.com/eafer/rdrview
- [4] M. Hoftich. *The Linebreaker package*. Prevent overflow boxes with LuaL^AT_EX. ctan.org/pkg/linebreaker
- [5] M. Hoftich. *The Luavlna package*. Prevent line breaks after single letter words, units, or academic titles. ctan.org/pkg/luavlna
- [6] M. Hoftich. *The LuaXML package*. Lua library for reading and serialising XML files. ctan.org/pkg/luaxml
- [7] M. Hoftich. *The Responsive package*. Responsive design methods for L^AT_EX. ctan.org/pkg/responsive
- [8] M. Hoftich. *Rmodepdf*. Convert web pages in reader mode to PDF. github.com/michal-h21/rmodepdf
- [9] S. Mortensen. The typographic scale, 2011. spencermortensen.com/articles/typographic-scale/

◇ Michal Hoftich
Magdalény Rettigové 4
Praha, 116 39
Czechia
[michal.h21 \(at\) gmail dot com](mailto:michal.h21@gmail.com)
<https://www.kodymirus.cz/>