I extracted text out of  
Markdown-formatted threads with regular expressions. I basically  
hammered away at the issues using tools I was familiar with until it  
worked! Now I know there’s a much better and cleaner way, that I’ll  
present in this note. Read on if you want to extract insights about  
text, code, links, etc. from R Markdown reports, Hugo website sources,  
GitHub issues… without writing messy and smelly code!

**Introduction to Markdown rendering and parsing**

This note will appear to you, dear reader, as an html page, either here  
on ropensci.org or on R-Bloggers, but I’m writing it as an R Markdown  
document, using Markdown syntax. I’ll knit it to Markdown and then  
Hugo’s Markdown processor, Blackfriday package, will transform  
it to html. Elements such as # blabla thus get transformed to

The rendering of Markdown to html or XML can also be used as a way to  
*parse* it, spelling package of R Markdown files, before spell checking them only. I had an  
aha moment when seeing this spelling strategy: why did I ever use  
regex to parse Markdown for text analysis?! Transforming it to XML  
first, and then using XPath, would be much cleaner!

It’s that feeling when you want to do something that sounds simple but  
instead your code is like 10 stack overflow snippets slapped together  
that you could never explain to another human what they do   
[pic.twitter.com/IF53AX6QvC](https://t.co/IF53AX6QvC)

— Dr. Alison Hill (@apreshill)  
[31  
d’agost de 2018](https://twitter.com/apreshill/status/1035526182392000514?ref_src=twsrc%5Etfw)

**From Markdown to XML**

In this note I’ll use my local fork of rOpenSci’s website source, and  
use all the Markdown sources of blog posts as example data. The chunk  
below is therefore not portable, sorry about that.

roblog <- "C:\\Users\\Maelle\\Documents\\ropensci\\roweb2\\content\\blog"

all\_posts <- fs::dir\_ls(roblog, regexp = "\*.md")

all\_posts <- all\_posts[all\_posts != "\_index.md"]

My fork master branch isn’t entirely synced. It has 202 posts.

The code below uses the commonmark package to render Markdown to  
XML. Commonmark is a standardized specification for Markdown syntax by  
[John McFarlane](https://johnmacfarlane.net/tools). The commonmark R  
package by Jeroen Ooms wraps the official  
cmark library. Note that my function still  
has a hacky element, it uses a blogdown unexported function to strip  
the YAML header of posts!.

library("magrittr")

get\_one\_xml <- function(md){

md %>%

readLines(encoding = "UTF-8") %>%

blogdown:::split\_yaml\_body() %>%

.$body %>%

commonmark::markdown\_xml(extensions = TRUE) %>%

xml2::read\_xml()

}

See what it gives me for one post.

get\_one\_xml(all\_posts[42])

## {xml\_document}

##

## [1] \n We just released a new version of \n < ...

## [2] \n First, install and load taxize\ ...

## [3] install.packages("rgbif")\n

## [4] library(taxize)\n

## [5] \n New things\n

## [6] \n New functions: class2tree\n\n Sometimes you just want to have a visual of th ...

## [8] \n Define a species list\n

## [9] spnames <- c("Latania lontaroides", "Randia ...

## [10] \n Then collect taxonomic hierarchies for each ta ...

## [11] out <- classification(spnames, db = "ncbi", ...

## [12] \n Use \n class2tree\n tr <- class2tree(out)\nplot(tr, no.margin = ...

## [14] \n \n New functions: get\_gbfid\n\n The Global Biodiversity Information Facility ( ...

## [17] \n We added a similar function to our \n ...

## [18] get\_gbifid(sciname = "Poa annua", verbose = FA ...

## [19] ## 1\n## "2704179"\n## attr(,"class")\n## [1] " ...

## [20] get\_gbifid(sciname = "Pinus contorta", verbose ...

## ...

Headings, code blocks… all properly delimited and one XPath query away  
from us! Let me convert all posts before diving into parsing examples.

all\_posts %>%

purrr::map(get\_one\_xml) -> blog\_xml

**Parsing the XML**

**URLs parsing**

Let’s say I want to find out which domains are the most often linked  
from rOpenSci’s blog. No need for any regular expression thanks to  
commonmark, XML2 and urltools!

get\_urls <- function(post\_xml){

post\_xml %>%

xml2::xml\_find\_all(xpath = './/d1:link', xml2::xml\_ns(post\_xml)) %>%

xml2::xml\_attr("destination") %>%

urltools::url\_parse()

}

# URLs

blog\_xml %>%

purrr::map\_df(get\_urls) %>%

dplyr::count(domain, sort = TRUE) %>%

head(n = 10) %>%

knitr::kable()

| **domain** | **n** |
| --- | --- |
| github.com | 1111 |
| ropensci.org | 272 |
| twitter.com | 167 |
| cran.r-project.org | 130 |
| en.wikipedia.org | 60 |
| ropensci.github.io | 29 |
| doi.org | 27 |
| bioconductor.org | 15 |
| unconf17.ropensci.org | 15 |
| www.gbif.org | 15 |

More Twitter than CRAN! We probably could do with less own-domain use  
since / would get us here too.

**R code parsing**

get\_functions <- function(post\_xml){

post\_xml %>%

# select all code chunks

xml2::xml\_find\_all(xpath = './/d1:code\_block', xml2::xml\_ns(.)) %>%

# select chunks with language info

.[xml2::xml\_has\_attr(., "info")] %>%

# select R chunks

.[xml2::xml\_attr(., "info") == "r"] %>%

# get the content of these chunks

xml2::xml\_text() %>%

glue::glue\_collapse(sep = "\n") -> code\_text

# Base R code parsing tools

parsed\_code <- try(parse(text = code\_text,

keep.source = TRUE) %>%

utils::getParseData(),

silent = TRUE)

if(is(parsed\_code, "try-error")){

# this happens because of output sometimes

# stored in R chunks when not using R Markdown

return(NULL)

}

if(is.null(parsed\_code)){

return(NULL)

}

dplyr::filter(parsed\_code,

grepl("FUNCTION", token))

}

blog\_xml %>%

purrr::map\_df(get\_functions) %>%

dplyr::count(text, sort = TRUE) %>%

head(n = 10) %>%

knitr::kable()

| **text** | **n** |
| --- | --- |
| library | 263 |
| c | 210 |
| aes | 106 |
| filter | 71 |
| mutate | 64 |
| ggplot | 58 |
| function | 53 |
| install.packages | 50 |
| install\_github | 38 |
| select | 38 |

Function definititions (function), basic stuff (c, library) and  
tidyverse functions seem to be the most popular on the blog!

**Text parsing**

After complementing our commonmark–xml2 combo with urltools and  
with R base code parsing facilities… let’s pair it with tidytext! What are the words most  
commonly use d n rOpenSci’s blog posts?

get\_text <- function(post\_xml){

xml2::xml\_find\_all(post\_xml,

xpath = './/d1:text', xml2::xml\_ns(post\_xml)) %>%

xml2::xml\_text(trim = TRUE) %>%

glue::glue\_collapse(sep = " ") %>%

as.character() -> text

tibble::tibble(text = text)

}

blog\_xml %>%

purrr::map\_df(get\_text) %>%

tidytext::unnest\_tokens(word, text, token = "words") %>%

dplyr::filter(!word %in% tidytext::stop\_words$word) %>%

dplyr::count(word, sort = TRUE) %>%

head(n = 10) %>%

knitr::kable()

| **word** | **n** |
| --- | --- |
| data | 1969 |
| package | 1097 |
| ropensci | 569 |
| packages | 486 |
| time | 412 |
| community | 394 |
| code | 377 |
| github | 358 |
| software | 302 |
| science | 297 |

This beats my old code! There’s really something to be said for  
purpose-built tools.

**Conclusion**

I hope this note will inspire you to use commonmark and xml2 when  
analyzing Markdown files. The results of XML-parsing are also better

parsed without (your writing)  
regular expressions: I have shown urltools for URL parsing, that base  
R has code parsing tools (parse, getParsedData), and I’ve used  
tidytext.