The Feedly category I have setup for git-stalking has indicated a fairly massive interest The Art of the Command Line. What is “The Art of the Command Line”? To quote the author(s):

Fluency on the command line is a skill often neglected or considered arcane, but it improves your flexibility and productivity as an engineer in both obvious and subtle ways. This is a selection of notes and tips on using the command-line that we’ve found useful when working on Linux. Some tips are elementary, and some are fairly specific, sophisticated, or obscure. This page is not long, but if you can use and recall all the items here, you know a lot.

It’s a great resource just the way it is (simple, plain markdown rendered in GitUgh). But, we can make it even greater with some help from rmarkdown::render() and some content slicing & dicing.

My initial thought was to grab the English version, put an R Markdown YAML header on it, remove some intro cruft and render it to standalone HTML. While that would be quick, easy and useful it’s also very manual and brittle since updating it would require copy and paste; plus, it leaves out the translated versions.

So, goal number uno became “make a function to do this”. Then, I realized *“Hey! This is a resource for command line stuff so why not turn the function into a command line tool!”*. So this became goal number II. (I have an internal posit that R adoption would be much higher if there were more easy-to-install command line utilities built in R since that’s one reason Python has a larger install base and many folks end up just using the command line versions of modules they install. CRAN’s draconian rules on what you can do during a package install makes this somewhat moot, tho. One could argue that CRAN is doing the right thing and that Python/PyPI are woefully insecure-by-default which is also true.)

**Goal Uno**

Since we’re going to create a function it also makes sense to parameterize options for the language, doc-theme and highlight-theme.

The setup plan for this is endeavour is pretty straightforward:

* fetch the current set of translations available
* check to make sure the desired translation is in ^^ set
* grab a copy of the specified document
* get the title (since that’s translated for each)
* remove some unnecessary front-matter
* turn the AUTHORS.md link into a proper link (vs relative)
* add in the YAML header with the desired customizations
* render the document to standalone HTML
* optionally open it after render

And, this is what that looks like:

taotcl <- function(language = "", theme = "simplex", highlight = "espresso", output\_dir = getwd(), open = TRUE) {

language <- language[1]

# find translations

httr::GET(

url = "https://api.github.com/repos/jlevy/the-art-of-command-line/contents/",

httr::add\_headers(

`Accept` = "application/vnd.github.v3+json"

),

httr::user\_agent("taotcl R script; @hrbrmstr")

) -> res

httr::stop\_for\_status(res)

ls <- httr::content(res, as = "parsed")

readmes <- Filter(function(.x) grepl("^README", .x), vapply(ls, `[[`, character(1), "name"))

langs <- regmatches(readmes, regexpr("-[-[:alpha:]]+", readmes))

# check to make sure a valid one was specified

if (language != "") { # "" => English

language <- sprintf("-%s", language)

if (!(language %in% langs)) {

stop(

"Language '", sub("^-", "", language),

"' not found in repo. Current translations include: ",

paste0(sprintf("'%s'", sub("^-", "", langs)), collapse = ", "),

".", call.=FALSE

)

}

}

# get the desired doc

src <- "https://raw.githubusercontent.com/jlevy/the-art-of-command-line/master/README{language}.md"

src <- glue::glue(src)

l <- readLines(src)

# find the title

title <- sub("^#[[:space:]]\*", "", l[which(grepl("^#[[:space:]]\*", l))[1]])

# figure out the cut line

cowsay <- which(grepl("cowsay", l))[1]

l <- l[-(1:(cowsay+1))] # cut

# make the AUTHORS.md a useful link

l <- gsub("(AUTHORS.md)", "(https://github.com/jlevy/the-art-of-command-line/blob/master/AUTHORS.md)", l, fixed = TRUE)

theme <- theme[1]

highlight <- highlight[1]

'---

title: "{title}"

author: "Joshua Levy"

email: "joshua@cal.berkeley.edu"

output:

html\_document:

theme: {theme}

highlight: {highlight}

toc: true

toc\_float: true

toc\_depth: 2

---

' -> yaml

# fill in the YAML

yaml <- glue::glue(yaml)

tf <- tempfile(fileext = ".Rmd")

on.exit(unlink(tf), add = TRUE)

writeLines(c(yaml, l), tf)

# render the doc

rmarkdown::render(

input = tf,

output\_file = sprintf("%s.html", tolower(gsub(" ", "-", title))),

output\_dir = output\_dir[1],

quiet = TRUE

) -> loc

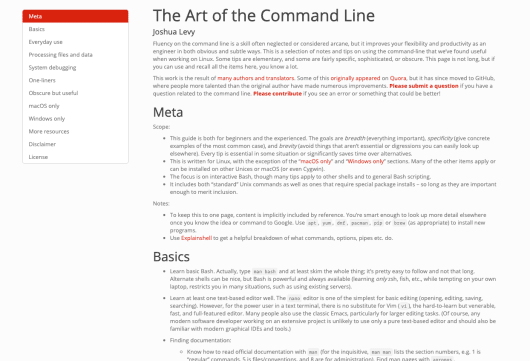
# open in browser

if (open[1]) browseURL(loc)

message("Rendered version is at '", loc, "'")

}

Running it with the defaults will have it look like this:

[](https://i2.wp.com/rud.is/b/wp-content/uploads/2019/05/render.png?ssl=1)

You don’t have to type it all as that function is in the taotcl.R script here:

usage: ./taotcl.R [--help] [--just-render] [--opts OPTS] [--language LANGUAGE] [--theme THEME] [--highlight HIGHLIGHT] [--output-dir OUTPUT-DIR]

or (on Windows): Rscript taotcl.R [--help] [--just-render] [--opts OPTS] [--language LANGUAGE] [--theme THEME] [--highlight HIGHLIGHT] [--output-dir OUTPUT-DIR]

Render 'The Art of the Command Line' to HTML

flags:

-h, --help show this help message and exit

-j, --just-render Only render the document. Do not open in the system default browser.

(Default is to render and open.)

optional arguments:

-x, --opts OPTS RDS file containing argument values

-l, --language LANGUAGE Language to render. Leave unspecified for English.

Current known: "cs", "de", "el", "es", "fr", "id", "it", "ja",

"ko", "pt", "ro", "ru", "sl", "uk", "zh-Hant", "zh" [default: ]

-t, --theme THEME Which R Markdown document theme to use. Ref: https://l.rud.is/2JOibrZ [default: simplex]

-c, --highlight HIGHLIGHT Which R Markdown code higlight theme to use. Ref: https://l.rud.is/2JOibrZ [default: espresso]

-o, --output-dir OUTPUT-DIR Where to store the rendered file. Defaults to current working directory.

[default: /your/current/directory/here]

**Goal II**

Now that we have a function we can call from R we just need a wrapper around it. I kinda like way David Shih put together his {argparser} package (it’s on CRAN) so we’ll make a wrapper for our rendering function with it.

We have pretty much the same goal list as the function in that we want to let users specify customizations. There are some additional ones as well (this is not an exhaustive list but it was “just enough” for this go):

1. Make it easy for folks on real operating systems to use it without the need to use Rscript
2. Let folks know what required packages they need to install if any are missing
3. Be quiet when loading packages
4. Assume friendly/useful defaults
5. Provide long and short parameters (some folks like short, some like long)

The first few lines of the finished script will accomplish #1-3:

#!/usr/bin/env Rscript

needed <- c("magrittr", "argparser", "httr", "glue", "rmarkdown")

installed <- rownames(installed.packages())

missing <- needed[!(needed %in% installed)]

if (length(missing)) stop("Please install the following packages: ", paste0(sprintf("'%s'", missing), collapse = ", "), call.=FALSE)

suppressPackageStartupMessages({

for (pkg in needed) {

require(package = pkg, quietly = TRUE, warn.conflicts = FALSE, character.only = TRUE)

}

})

We need a way to get command line parameters in, hence the use of {argparser}. We’ll create an arg\_parser object and then add arguments using the {magrittr} pipe (%>%). You can add long/short argument names as well as help and defaults (plus note whether an argument is a flag/toggle). Once we have those setup, we tell {argparser} to process any arguments provided by the user:

arg\_parser(

description = "Render 'The Art of the Command Line' to HTML"

) %>%

add\_argument(

arg = "--language",

help = 'Language to render. Leave unspecified for English. Current known: "cs", "de", "el", "es", "fr", "id", "it", "ja", "ko", "pt", "ro", "ru", "sl", "uk", "zh-Hant", "zh"',

type = "character",

short = "-l",

default = ""

) %>%

add\_argument(

arg = "--theme",

help = "Which R Markdown document theme to use. Ref: https://l.rud.is/2JOibrZ",

type = "character",

short = "-t",

default = "simplex"

) %>%

add\_argument(

arg = "--highlight",

help = "Which R Markdown code higlight theme to use. Ref: https://l.rud.is/2JOibrZ",

type = "character",

short = "-c",

default = "espresso"

) %>%

add\_argument(

arg = "--output-dir",

help = "Where to store the rendered file. Defaults to current working directory.",

type = "character",

short = "-o",

default = getwd()

) %>%

add\_argument(

arg = "--just-render",

help = "Only render the document. Do not open in the system default browser. (Default is to render and open.)",

short = "-j",

flag = TRUE

) -> parser

opts <- argparser::parse\_args(parser)

Once we have those (the taotcl() function would come next in the source) then it’s just a matter of calling the function:

taotcl(

language = opts$language,

theme = opts$theme,

highlight = opts$highlight,

output\_dir = opts$output\_dir,

open = is.na(opts$just\_render) | (!opts$just\_render)

)

If the command line program we’ve just made is called with a -h or --help the user will get:

usage: ./taotcl.R [--help] [--just-render] [--opts OPTS] [--language LANGUAGE] [--theme THEME] [--highlight HIGHLIGHT] [--output-dir OUTPUT-DIR]

or (on Windows): Rscript taotcl.R [--help] [--just-render] [--opts OPTS] [--language LANGUAGE] [--theme THEME] [--highlight HIGHLIGHT] [--output-dir OUTPUT-DIR]

Render 'The Art of the Command Line' to HTML

flags:

-h, --help show this help message and exit

-j, --just-render Only render the document. Do not open in the system default browser.

(Default is to render and open.)

optional arguments:

-x, --opts OPTS RDS file containing argument values

-l, --language LANGUAGE Language to render. Leave unspecified for English.

Current known: "cs", "de", "el", "es", "fr", "id", "it", "ja",

"ko", "pt", "ro", "ru", "sl", "uk", "zh-Hant", "zh" [default: ]

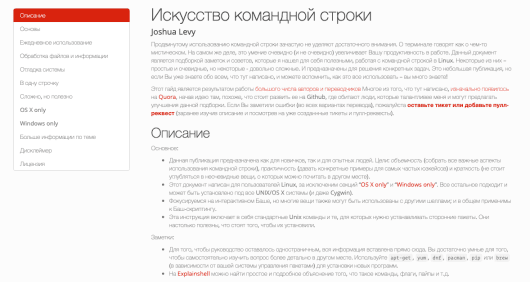
-t, --theme THEME Which R Markdown document theme to use. Ref: https://l.rud.is/2JOibrZ [default: simplex]

-c, --highlight HIGHLIGHT Which R Markdown code higlight theme to use. Ref: https://l.rud.is/2JOibrZ [default: espresso]

-o, --output-dir OUTPUT-DIR Where to store the rendered file. Defaults to current working directory.

[default: /your/current/directory/here]

If we run it with, say, ./taotcl.R --language ru -o /tmp the script will process the correct language version and render it to /tmp/искусство-командной-строки.html plus auto-open it for us. It should look like:

[](https://i2.wp.com/rud.is/b/wp-content/uploads/2019/05/render-ru.png?ssl=1)