Project Management

Matthew Leonawicz

January 7, 2015

1 Introduction

This is a project management project. While current projects are shown in the chart below, the aim of this project is the development of convenient **R**-related project management tools.

1.1 Motivation

I am working on these tools to enhance my own workflow across multiple **R** projects.

1.2 Details

R code for the project will be compiled into an **R** package, `rpm` for easy use. This is a personal package and will not be available anywhere but my github repository, but you are welcome to explore the package and functions. It is unlikely that you would manage your **R** projects in the same manner that I do, but if you do, or just want some ideas, feel free to explore.

1.2.1 Capabilities

`rpm` can create a new project. This essentially generates a specific directory structure which I use often to manage project files. For an existing project, once **R** scripts have been created, `rpm` can generate template Rmd files for each. For existing Rmd files, `rpm` can conveniently append these **R** Markdown files with a list of any new `knitr` code chunk names in project **R** scripts being developed which have not yet been included in the respective Rmd files.

1.2.2 Limitations

While `rpm` assists with project documentation, this mainly takes the form of file generation and appending. Documentation is unique to every project of course. Every script is different. The most that is possible is to auto-fill commonly used code chunk names and metadata. Each document must be written individually by the author, but when a project has many **R** scripts requiring documentation, it is nice to not have to create all the corresponding Rmd files by hand and copy and paste generic contents.

The project management code is not yet in package form. Many additional features are yet to be incorporated. Generic code relating to the further processing of Rmd files into various other output files via `rmarkdown` and `knitr` remains at an early development stage.

2 Related items

Currently there is only this unpackaged **R** script, accompanying code for a projects hierarchy diagram and a code flow diagram based on the current development of this project, and a simple script for generating documents based on project **R** code.

2.1 Files and Data

This project does not use any data. It does make use of supplemental libraries for formatting during html document generation. `proj_sankey.R` and `code_sankey.R` are used to produce of project hierarchy diagram of my current projects and a code flow diagram for this project, respectively. `drg.R` is used to assist in dynamic report generation.

2.2 Code flow

The Sankey diagram has become part of project management. Each project has its own, detailing the relationships among **R** code and data relevant to the project, and in some cases, how they relate to code and data files which are more general and span multiple projects. In general, for my projects I would only provide the code flow diagram here among the rest of the project documentation, but since this is the project management project and I am introducing its use, in this case I will also show the [code](code_sankey.html "code") I use to make the diagram.

2.3 Current projects

Here is a project hierarchy diagram showing the relationships among all my current projects.

3 R code

3.1 Template objects

Character string objects are defined which are used to fill templates when generating new files for a project. A tentative default path is also included since this code relates to my own work.

```
# For package 'rpm'

# data

rmd.template <- "\n\n## Introduction\nADD_TEXT_HERE\n\n### Motivation\nADD_TEXT_HERE\n\n### Details\nADD

# default path
matt.proj.path <- "C:/github"
```

3.2 Package functions

Functions are defined for creating new projects, generating Rmd files for project **R** scripts, and appending these **R** Markdown files with updated information from the corresponding **R** scripts as their development continues. Additional functions will be incorporated later.

3.2.1 newProject

`newProject` creates a new named project directory structure at the specified file path. If a directory with this project name already exists in this location on the file system and `overwrite=FALSE`, the function will abort. Default project subdirectories are created unless a different vector of folder names is explicitly passed to `newProject`. If one of the subdirectories is `docs` then the default vector of subdirectories under `docs` is also created. This argument can also be set explicitly. The current function only creates directories, not files, so `overwrite=TRUE` is safe to use on any existing project.

```
newProject <- function(name, path, dirs = c("code", "data", "docs", "plots",
    "workspaces"), docs.dirs = c("diagrams", "ioslides", "notebook", "Rmd/include",
    "md", "html", "Rnw", "pdf", "timeline", "tufte"), overwrite = FALSE) {
```

```
    stopifnot(is.character(name))
    name <- file.path(path, name)
    if (file.exists(name) && !overwrite)
        stop("This project already exists.")
    dir.create(name, recursive = TRUE, showWarnings = FALSE)
    if (!file.exists(name))
        stop("Directory appears invalid.")

    path.dirs <- file.path(name, dirs)
    sapply(path.dirs, dir.create, showWarnings = FALSE)
    path.docs <- file.path(name, "docs", docs.dirs)
    if ("docs" %in% dirs)
        sapply(path.docs, dir.create, recursive = TRUE, showWarnings = FALSE)
    if (overwrite)
        cat("Project directories updated.\n") else cat("Project directories created.\n")
}
```

3.2.2 rmdHeader

`rmdHeader` generates the yaml metadata header for Rmd files as a character string to be inserted at the top of a file. It has default arguments specific to my own projects but can be changed. The output from this function is passed directly to `genRmd` below.

```
rmdHeader <- function(title = "INSERT_TITLE_HERE", author = "Matthew Leonawicz",
    theme = "united", highlight = "zenburn", toc = FALSE, keep.md = TRUE, ioslides = FALSE,
    include.pdf = FALSE) {
    if (toc)
        toc <- "true" else toc <- "false"
    if (keep.md)
        keep.md <- "true" else keep.md <- "false"
    if (ioslides)
        hdoc <- "ioslides_presentation" else hdoc <- "html_document"
    rmd.header <- "---\n"
    if (!is.null(title))
        rmd.header <- paste0(rmd.header, "title: ", title, "\n")
    if (!is.null(author))
        rmd.header <- paste0(rmd.header, "author: ", author, "\n")
    rmd.header <- paste0(rmd.header, "output:\n  ", hdoc, ":\n    toc: ", toc,
        "\n    theme: ", theme, "\n    highlight: ", highlight, "\n    keep_md: ",
        keep.md, "\n")
    if (ioslides)
        rmd.header <- paste0(rmd.header, "    widescreen: true\n")
    if (include.pdf)
        rmd.header <- paste0(rmd.header, "  pdf_document:\n    toc: ", toc,
            "\n    highlight: ", highlight, "\n")
    rmd.header <- paste0(rmd.header, "---\n")
    rmd.header
}
```

3.2.3 rmdknitrSetup

`rmdknitrSetup` generates the `knitr` global options setup code cunk for Rmd files as a character string to be inserted at the top of a file following the yaml header. The only option at this time is the ability to include or

exclude a source reference to a project-related code flow diagram **R** script via `include.sankey`. The output from this function is passed directly to `genRmd` below.

```
rmdknitrSetup <- function(file, include.sankey = TRUE) {
    x <- paste0("\n```{r knitr_setup, echo=FALSE}\nopts_chunk$set(cache=FALSE, eval=FALSE, tidy=TRUE, me
    if (include.sankey)
        x <- paste0(x, "read_chunk(\"../../code/proj_sankey.R\")\n")
    x <- paste0(x, "read_chunk(\"../../code/", file, "\")\n```\n")
    x
}
```

3.2.4 genRmd

`genRmd` works on existing projects. It checks for existing **R** scripts. If no **R** files exist in the project's `code` directory, the function will abort. Otherwise it will generate Rmd template files for each of the **R** scripts it finds.

With `replace=TRUE` any existing Rmd files are regenerated with the provided template - be careful! With `replace=FALSE` (default) Rmd files are generated only for **R** scripts which do not yet have corresponding Rmd files. If `update.header=TRUE`, `replace` is ignored, and only existing Rmd files are regenerated, in this case strictly updating the yaml metadata header at the top of each Rmd file without altering any other Rmd content/documentation.

The Rmd files are placed in the `/docs/Rmd` directory. This function assumes this project directory exists.

```
genRmd <- function(path, replace = FALSE, header = rmdHeader(), knitrSetupChunk = rmdknitrSetup(),
    update.header = FALSE, ...) {
    stopifnot(is.character(path))
    files <- list.files(path, pattern = ".R$", full = TRUE)
    stopifnot(length(files) > 0)
    rmd <- gsub(".R", ".Rmd", basename(files))
    rmd <- file.path(dirname(path), "docs/Rmd", rmd)
    if (!(replace | update.header))
        rmd <- rmd[!sapply(rmd, file.exists)]
    if (update.header)
        rmd <- rmd[sapply(rmd, file.exists)]
    stopifnot(length(rmd) > 0)

    sinkRmd <- function(x, ...) {
        y1 <- header
        y2 <- kniterSetupChunk
        y3 <- list(...)$rmd.template
        if (is.null(y1))
            y1 <- rmd.header
        if (is.null(y2))
            y2 <- rmd.knitr.setup(gsub(".Rmd", ".R", basename(x)))
        if (is.null(y3))
            y3 <- rmd.template
        sink(x)
        sapply(c(y1, y2, y3), cat)
        sink()
    }

    swapHeader <- function(x) {
        l <- readLines(x)
```

```
        ind <- which(l == "---")
        l <- l[(ind[2] + 1):length(l)]
        l <- paste0(l, "\n")
        sink(x)
        sapply(c(header, l), cat)
        sink()
    }

    if (update.header) {
        sapply(rmd, swapHeader, ...)
        cat("yaml header updated for each .Rmd file.\n")
    } else {
        sapply(rmd, sinkRmd, ...)
        cat(".Rmd files created for each .R file.\n")
    }
}
```

3.2.5 chunkNames

`chunkNames` can be used in two ways. It can return a list with length equal to the number of **R** files, where each list element is a vector of **R** code chunk names found in each **R** script.

Alternatively, with `append.new=TRUE`, this list has each vector matched element-wise against chunk names found in existing Rmd files. If no Rmd files have yet been generated, the function will abort. Otherwise, for the Rmd files which do exist (and this may correspond to a subset of the **R** files), these Rmd files are appended with a list of code chunk names found in the current corresponding **R** files which have not yet been integrated into the current state of the Rmd files. This fascilitates updating of Rmd documentation when it falls behind scripts which have been updated.

```
chunkNames <- function(path, rChunkID = "# @knitr", rmdChunkID = "```{r", append.new = FALSE) {
    files <- list.files(path, pattern = ".R$", full = TRUE)
    stopifnot(length(files) > 0)
    l1 <- lapply(files, readLines)
    l1 <- rapply(l1, function(x) x[substr(x, 1, nchar(rChunkID)) == rChunkID],
        how = "replace")
    l1 <- rapply(l1, function(x, p) gsub(paste0(p, " "), "", x), how = "replace",
        p = rChunkID)
    if (!append.new)
        return(l1)

    appendRmd <- function(x, rmd.files, rChunks, rmdChunks, ID) {
        r1 <- rmdChunks[[x]]
        r2 <- rChunks[[x]]
        r.new <- r2[!(r2 %in% r1)]
        if (length(r.new)) {
            r.new <- paste0(ID, " ", r.new, "}\n```\n", collapse = "")  # Hard coded brace and backticks
            sink(rmd.files[x], append = TRUE)
            cat("\nNEW_CODE_CHUNKS\n")
            cat(r.new)
            sink()
            paste(basename(rmd.files[x]), "appended with new chunk names from .R file")
        } else paste("No new chunk names appended to", basename(rmd.files[x]))
    }
```

```
    rmd <- gsub(".R", ".Rmd", basename(files))
    rmd <- file.path(dirname(path), "docs/Rmd", rmd)
    rmd <- rmd[sapply(rmd, file.exists)]
    stopifnot(length(rmd) > 0)  # Rmd files must exist
    files.ind <- match(gsub(".Rmd", "", basename(rmd)), gsub(".R", "", basename(files)))  # Rmd exist fo
    l2 <- lapply(rmd, readLines)
    l2 <- rapply(l2, function(x) x[substr(x, 1, nchar(rmdChunkID)) == rmdChunkID],
        how = "replace")
    l2 <- rapply(l2, function(x, p) gsub(paste0(p, " "), "", x), how = "replace",
        p = gsub("\\{", "\\\\{", rmdChunkID))
    l2 <- rapply(l2, function(x) gsub("}", "", sapply(strsplit(x, ","), "[[",
        1)), how = "replace")
    sapply(1:length(rmd), appendRmd, rmd.files = rmd, rChunks = l1[files.ind],
        rmdChunks = l2, ID = rmdChunkID)
}
```

Regarding the creation and updating of Rmd files, `rpm` simply assumes that there will be one **R** Markdown file corresponding to one **R** script. This is not always the case for a given project, but again, the purpose is to generate basic templates. Unnecessary files can always be deleted later, or edits made such that one **R** Markdown file reads multiple **R** scripts, as is the case with the Rmd file used to generate this document.

The main function for conversion between Rmd and Rnw files is `convertDocs`. This function contains several internal support functions, each of which is somewhat limited in how much specific conversion it can achieve. The functions below were written with my particular style of Rmd and Rnw documentation in mind. As such they are necessarily a bit idiosyncratic and cannot account for every possible difference found between Rmd and Rnw formatting across any pair of documents. I only strived to speed up the process by which I convert my own documents, most of which follow a set of general rules and expectations most of the time. Anything atypical which doesn't convert properly can be adjusted by hand afterward. This is still better than rewriting, copy-pasting, and search-and-replacing many sections of many files on a recurring basis. Further improvements in conversion will be added later.

3.2.6 .swapHeadings

`.swapHeadings` assists in bidirectional conversion between Rmd and Rnw files. It swaps section headings formatting. It is called directly by `swap`, internal to `convertDocs`.

```
# Rmd <-> Rnw document conversion Conversion support functions called by
# .swap()
.swapHeadings <- function(from, to, x) {
    nc <- nchar(x)
    ind <- which(substr(x, 1, 1) == "\\")
    if (!length(ind)) {
        # assume Rmd file
        ind <- which(substr(x, 1, 1) == "#")
        ind.n <- rep(1, length(ind))
        for (i in 2:6) {
            ind.tmp <- which(substr(x[ind], 1, i) == substr("######", 1, i))
            if (length(ind.tmp))
                ind.n[ind.tmp] <- ind.n[ind.tmp] + 1 else break
        }
        for (i in 1:length(ind)) {
            n <- ind.n[i]
            input <- paste0(substr("######", 1, n), " ")
            h <- x[ind[i]]
            h <- gsub("\\*", "_", h)  # Switch any markdown boldface asterisks in headings to double und
```

```
            heading <- gsub("\n", "", substr(h, n + 2, nc[ind[i]]))
            # h <- gsub(input, '', h)
            if (n <= 2)
                subs <- "\\" else if (n == 3)
                subs <- "\\sub" else if (n == 4)
                subs <- "\\subsub" else if (n >= 5)
                subs <- "\\subsubsub"
            output <- paste0("\\", subs, "section{", heading, "}\n")
            x[ind[i]] <- gsub(h, output, h)
        }
    } else {
        # assume Rnw file
        ind <- which(substr(x, 1, 8) == "\\section")
        if (length(ind)) {
            for (i in 1:length(ind)) {
                h <- x[ind[i]]
                heading <- paste0("## ", substr(h, 10, nchar(h) - 2), "\n")
                x[ind[i]] <- heading
            }
        }
        ind <- which(substr(x, 1, 4) == "\\sub")
        if (length(ind)) {
            for (i in 1:length(ind)) {
                h <- x[ind[i]]
                z <- substr(h, 2, 10)
                if (z == "subsubsub") {
                  p <- "##### "
                  n <- 19
                } else if (substr(z, 1, 6) == "subsub") {
                  p <- "#### "
                  n <- 16
                } else if (substr(z, 1, 3) == "sub") {
                  p <- "### "
                  n <- 13
                }
                heading <- paste0(p, substr(h, n, nchar(h) - 2), "\n")
                x[ind[i]] <- heading
            }
        }
    }
    x
}
```

3.2.7 .swapChunks

`.swapChunks` assists in bidirectional conversion between Rmd and Rnw files. It swaps code chunk formatting. It is called directly by `swap`, internal to `convertDocs`.

```
# Rmd <-> Rnw document conversion Conversion support functions called by
# .swap()
.swapChunks <- function(from, to, x, offset.end = 1) {
    gsbraces <- function(txt) gsub("\\{", "\\\\{", txt)
    nc <- nchar(x)
```

```
    chunk.start.open <- substr(x, 1, nchar(from[1])) == from[1]
    chunk.start.close <- substr(x, nc - offset.end - nchar(from[2]) + 1, nc -
        offset.end) == from[2]
    chunk.start <- which(chunk.start.open & chunk.start.close)
    chunk.end <- which(substr(x, 1, nchar(from[3])) == from[3] & nc == nchar(from[3]) +
        offset.end)
    x[chunk.start] <- gsub(from[2], to[2], gsub(gsbraces(from[1]), gsbraces(to[1]),
        x[chunk.start]))
    x[chunk.end] <- gsub(from[3], to[3], x[chunk.end])
    chunklines <- as.numeric(unlist(mapply(seq, chunk.start, chunk.end)))
    list(x, chunklines)
}
```

3.2.8 .swapEmphasis

`.swapEmphasis` assists in bidirectional conversion between Rmd and Rnw files. It swaps text formatting such as boldface and typewriter font. It is called directly by `swap`, internal to `convertDocs`.

```
# Rmd <-> Rnw document conversion Conversion support functions called by
# .swap() I know I use '**' strictly for bold font in Rmd files.  For now,
# this function assumes: 1. The only emphasis in a doc is boldface or
# typewriter.  2. These instances are always preceded by a space, a carriage
# return, or an open bracket, 3. and followed by a space, period, comma, or
# closing bracket.
.swapEmphasis <- function(x, emphasis = "remove", pat.remove = c("`", "\\*\\*",
    "__"), pat.replace = pat.remove, replacement = c("\\\\texttt\\{", "\\\\textbf\\{",
    "\\\\textbf\\{", "\\}", "\\}", "\\}")) {

    stopifnot(emphasis %in% c("remove", "replace"))
    n <- length(pat.replace)
    rep1 <- replacement[1:n]
    rep2 <- replacement[(n + 1):(2 * n)]
    prefix <- c(" ", "^", "\\{", "\\(")
    suffix <- c(" ", ",", "-", "\n", "\\.", "\\}", "\\)")
    n.p <- length(prefix)
    n.s <- length(suffix)
    pat.replace <- c(paste0(rep(prefix, n), rep(pat.replace, each = n.p)), paste0(rep(pat.replace,
        each = n.s), rep(suffix, n)))
    replacement <- c(paste0(rep(gsub("\\^", "", prefix), n), rep(rep1, each = n.p)),
        paste0(rep(rep2, each = n.s), rep(suffix, n)))
    if (emphasis == "remove")
        for (k in 1:length(pat.remove)) x <- sapply(x, function(v, p, r) gsub(p,
            r, v), p = pat.remove[k], r = "")
    if (emphasis == "replace")
        for (k in 1:length(pat.replace)) x <- sapply(x, function(v, p, r) gsub(p,
            r, v), p = pat.replace[k], r = replacement[k])
    x
}
```

3.2.9 .swap

`.swap` assists in bidirectional conversion between Rmd and Rnw files. It is called internal to `convertDocs`.

```
# Rmd <-> Rnw document conversion Conversion support functions called by
# .convertDocs()
.swap <- function(file, header = NULL, outDir, rmdChunkID, rnwChunkID, emphasis,
    overwrite, ...) {
    title <- list(...)$title
    author <- list(...)$author
    highlight <- list(...)$highlight
    ext <- tail(strsplit(file, "\\.")[[1]], 1)
    l <- readLines(file)
    l <- l[substr(l, 1, 7) != "<style>"]  # Strip any html style lines
    if (ext == "Rmd") {
        from <- rmdChunkID
        to <- rnwChunkID
        hl.default <- "solarized-light"
        out.ext <- "Rnw"
        h.ind <- 1:which(l == "---")[2]
        h <- l[h.ind]
        t.ind <- which(substr(h, 1, 7) == "title: ")
        a.ind <- which(substr(h, 1, 8) == "author: ")
        highlight.ind <- which(substr(h, 1, 11) == "highlight: ")
        if (is.null(title) & length(t.ind))
            title <- substr(h[t.ind], 8, nchar(h[t.ind])) else if (is.null(title))
            title <- ""
        if (is.null(author) & length(a.ind))
            author <- substr(h[a.ind], 9, nchar(h[a.ind])) else if (is.null(author))
            author <- ""
        if (is.null(highlight) & length(highlight.ind))
            highlight <- substr(h[highlight.ind], 12, nchar(h[highlight.ind])) else if (is.null(highligh
            highlight <- hl.default else if (!(highlight %in% knit_theme$get()))
            highlight <- hl.default
        if (!is.null(title))
            header <- c(header, paste0("\\title{", title, "}"))
        if (!is.null(author))
            header <- c(header, paste0("\\author{", author, "}"))
        if (!is.null(title))
            header <- c(header, "\\maketitle\n")
        header <- c(header, paste0("<<highlight, echo=FALSE>>=\nknit_theme$set(knit_theme$get('",
            highlight, "'))\n@\n"))
    } else if (ext == "Rnw") {
        from <- rnwChunkID
        to <- rmdChunkID
        hl.default <- "tango"
        out.ext <- "Rmd"
        begin.doc <- which(l == "\\begin{document}")
        make.title <- which(l == "\\maketitle")
        if (length(make.title))
            h.ind <- 1:make.title else h.ind <- 1:begin.doc
        h <- l[h.ind]
        t.ind <- which(substr(h, 1, 6) == "\\title")
        a.ind <- which(substr(h, 1, 7) == "\\author")
        highlight.ind <- which(substr(l, 1, 11) == "<<highlight")
        if (is.null(title) & length(t.ind))
            title <- substr(h[t.ind], 8, nchar(h[t.ind]) - 1)
        if (is.null(author) & length(a.ind))
```

```
            author <- substr(h[a.ind], 9, nchar(h[a.ind]) - 1)
        if (length(highlight.ind)) {
            l1 <- l[highlight.ind + 1]
            h1 <- substr(l1, nchar("knit_theme$set(knit_theme$get('") + 1, nchar(l1) -
                nchar("'))\n"))
            if (!(h1 %in% knit_theme$get()))
                h1 <- hl.default
        }
        if (is.null(highlight) & length(highlight.ind))
            highlight <- h1 else if (is.null(highlight))
            highlight <- hl.default else if (!(highlight %in% knit_theme$get()))
            highlight <- hl.default
        header <- rmdHeader(title = title, author = author, highlight = highlight)
        h.chunks <- .swapChunks(from = from, to = to, x = h, offset.end = 0)
        header <- c(header, h.chunks[[1]][h.chunks[[2]]])
    }
    header <- paste0(header, collapse = "\n")
    l <- paste0(l[-h.ind], "\n")
    l <- .swapHeadings(from = from, to = to, x = l)
    chunks <- .swapChunks(from = from, to = to, x = l)
    l <- chunks[[1]]
    if (ext == "Rmd")
        l <- .swapEmphasis(x = l, emphasis = emphasis)
    if (ext == "Rmd")
        l[-chunks[[2]]] <- sapply(l[-chunks[[2]]], function(v, p, r) gsub(p,
            r, v), p = "_", r = "\\\\_")
    l <- c(header, l)
    if (ext == "Rmd")
        l <- c(l, "\n\\end{document}\n")
    if (ext == "Rnw") {
        ind <- which(substr(l, 1, 1) == "\\")  # drop any remaining lines beginning with a backslash
        l <- l[-ind]
    }
    outfile <- file.path(outDir, gsub(paste0("\\.", ext), paste0("\\.", out.ext),
        basename(file)))
    if (overwrite || !file.exists(outfile)) {
        sink(outfile)
        sapply(l, cat)
        sink()
        print(paste("Writing", outfile))
    }
}
```

3.2.10 convertDocs

`convertDocs` converts between Rmd and Rnw files. The project's `docs/Rmd` or `docs/Rnw` directory is specified. Any files of the same type as the directory are converted to the other type and saved to the other directory. The input files are not removed.

This function speeds up the process of duplicating files, e.g., when wanting to make PDFs from Rnw files when only Rmd files exist. This is almost exclusively what I use this function for. On less frequent occasions I have used it in the other direction when I have Rnw files which were once used to make PDFs but later I decide to put them on the web as a web page and not as a link to a PDF.

The user still makes specific changes by hand, for example, any required changes to `knitr` code chunk

options that must differ for PDF output vs. html output. The primary benefit is in not having to fuss with large amounts of standard substitutions which can be automated, such as swapping code chunk enclosure styles and common file metadata. Of course, this function is not perfect. It amounts to a text-parsing hack that is intended to handle the most common of idiosyncrasies and differences which exist between my own Rmd and Rnw files in the context of my own set of rules and assumptions, outlined below.

```
# Rmd <-> Rnw document conversion Main conversion function
convertDocs <- function(path, rmdChunkID = c("```{r", "}", "```"), rnwChunkID = c("<<",
    ">>=", "@"), emphasis = "replace", overwrite = FALSE, ...) {
    stopifnot(is.character(path))
    type <- basename(path)
    rmd.files <- list.files(path, pattern = ".Rmd$", full = TRUE)
    rnw.files <- list.files(path, pattern = ".Rnw$", full = TRUE)
    dots <- list(...)
    if (rmdChunkID[1] == "```{r")
        rmdChunkID[1] <- paste0(rmdChunkID[1], " ")
    if (type == "Rmd") {
        stopifnot(length(rmd.files) > 0)
        outDir <- file.path(dirname(path), "Rnw")
        if (is.null(doc.class <- dots$doc.class))
            doc.class <- "article"
        if (is.null(doc.packages <- dots$doc.packages))
            doc.packages <- "geometry"
        doc.class.string <- paste0("\\documentclass{", doc.class, "}")
        doc.packages.string <- paste0(sapply(doc.packages, function(x) paste0("\\usepackage{",
            x, "}")), collapse = "\n")
        if ("geometry" %in% doc.packages)
            doc.packages.string <- c(doc.packages.string, "\\geometry{verbose, tmargin=2.5cm, bmargin=2.
        header.rnw <- c(doc.class.string, doc.packages.string, "\\begin{document}\n")  #,
        # paste0('<<highlight, echo=FALSE>>=\nknit_theme$set(knit_theme$get('',
        # theme, ''))\n@\n'))
    } else if (type == "Rnw") {
        stopifnot(length(rnw.files) > 0)
        outDir <- file.path(dirname(path), "Rmd")
    } else stop("path must end in 'Rmd' or 'Rnw'.")
    if (type == "Rmd") {
        sapply(rmd.files, .swap, header = header.rnw, outDir = outDir, rmdChunkID = rmdChunkID,
            rnwChunkID = rnwChunkID, emphasis = emphasis, overwrite = overwrite,
            ...)
        cat(".Rmd to .Rnw file conversion complete.\n")
    } else {
        sapply(rnw.files, .swap, header = NULL, outDir = outDir, rmdChunkID = rmdChunkID,
            rnwChunkID = rnwChunkID, emphasis = emphasis, overwrite = overwrite,
            ...)
        cat(".Rnw to .Rmd file conversion complete.\n")
    }
}
```

3.2.11 moveDocs

`moveDocs` relocates files by renaming with a new file path. Specifically, it scans for md and html files in the `docs/Rmd` directory and/or pdf files in the `docs/Rnw` directory. If such files are found in the respective locations, they are moved to `docs/md`, `docs/html`, and `docs/pdf`, respectively.

The intent is to clean up the Rmd and Rnw directories after `knitr` has been used to knit documents in place. I do this because I have more success knitting documents with the confluence of `RStudio`, `rmarkdown`, `knitr`, `pandoc`, and `LaTeX` when the knitting occurs all within the directory of the originating files. The process is more prone to throwing errors when trying to specify alternate locations for outputs.

`moveDocs` makes a nominal effort to replace a possible relative path with a full file path before proceeding, if the former is supplied. Default arguments include `move=TRUE` which will call `file.rename` and `copy=FALSE` which, if `TRUE` (and `move=FALSE`), will alternatively call `file.copy`. If both are `TRUE`, any files found are moved.

This function will always overwrite any existing file versions previously moved to the output directories, by way of `file.rename`. To keep the behavior consistent, when `move=FALSE` and `copy=TRUE`, `file.copy` always executes with its argument, `overwrite=TRUE`. This should never cause problems because in the context I intend for this function, the types of files being moved or copied from `docs/Rmd` and `docs/Rnw` are never used as inputs to other files, functions, or processes, nor are they meant to be edited by hand after being generated.

If there are LaTeX-associated files present (.TeX, .aux, and .txt files with the same file names as local pdf files.), these files will be removed if `remove.latex=TRUE` (default). If `FALSE`, the default `latexDir="LaTeX"` means that these files will be moved to the `docs/LaTeX` directory rather than deleted. If this directory does not exist, it will be created. An alternate location can be specified, such as "pdf" if you want to keep these files with the related pdf files after those are moved by `moveDocs` as well to `docs/pdf`.

```
# Organization documentation
moveDocs <- function(path.docs, type = c("md", "html", "pdf"), move = TRUE,
    copy = FALSE, remove.latex = TRUE, latexDir = "latex") {
    if (any(!(type %in% c("md", "html", "pdf"))))
        stop("type must be among 'md', 'html', and 'pdf'")
    stopifnot(move | copy)
    if (path.docs == "." | path.docs == "./")
        path.docs <- getwd()
    if (strsplit(path.docs, "/")[[1]][1] == "..") {
        tmp <- strsplit(path.docs, "/")[[1]][-1]
        if (length(tmp))
            path.docs <- file.path(getwd(), paste0(tmp, collapse = "/")) else stop("Check path.docs argu
    }
    for (i in 1:length(type)) {
        if (type[i] == "pdf")
            origin <- "Rnw" else origin <- "Rmd"
        path.i <- file.path(path.docs, origin)
        infiles <- list.files(path.i, pattern = paste0("\\.", type[i], "$"),
            full = TRUE)
        if (type[i] == "pdf") {
            extensions <- c("tex", "aux", "log")
            all.pdfs <- basename(list.files(path.docs, pattern = ".pdf$", full = T,
                recursive = T))
            pat <- paste0("^", rep(gsub("pdf", "", all.pdfs), length(extensions)),
                rep(extensions, each = length(all.pdfs)), "$")
            latex.files <- unlist(sapply(1:length(pat), function(p, path, pat) list.files(path,
                pattern = pat[p], full = TRUE), path = path.i, pat = pat))
            print(latex.files)
            if (length(latex.files)) {
                if (remove.latex) {
                  unlink(latex.files)
                } else {
                  dir.create(file.path(path.docs, latexDir), showWarnings = FALSE,
                    recursive = TRUE)
```

```
                file.rename(latex.files, file.path(path.docs, latexDir, basename(latex.files)))
            }
        }
    }
    if (length(infiles)) {
        infiles <- infiles[basename(dirname(infiles)) == origin]
        if (length(infiles)) {
            if (type[i] == "html") {
              html.dirs <- gsub("\\.html", "_files", infiles)
              dirs <- list.dirs(path.i, recursive = FALSE)
              ind <- which(dirs %in% html.dirs)
              if (length(ind)) {
                html.dirs <- dirs[ind]
                html.dirs.recur <- list.dirs(html.dirs)
                for (p in 1:length(html.dirs.recur)) dir.create(gsub("/Rmd",
                  "/html", html.dirs.recur[p]), recursive = TRUE, showWarnings = FALSE)
                subfiles <- unique(unlist(lapply(1:length(html.dirs.recur),
                  function(p, path) list.files(path[p], full = TRUE), path = html.dirs.recur)))
                subfiles <- subfiles[!(subfiles %in% html.dirs.recur)]
                file.copy(subfiles, gsub("/Rmd", "/html", subfiles), overwrite = TRUE)
                if (move)
                  unlink(html.dirs, recursive = TRUE)
              }
            }
            outfiles <- file.path(path.docs, type[i], basename(infiles))
            if (move)
              file.rename(infiles, outfiles) else if (copy)
              file.copy(infiles, outfiles, overwrite = TRUE)
        }
    }
  }
}
```

3.2.12 genNavbar

`genNavbar` generates a navigation bar for a web page. The html file created is generally written to the project's `docs/Rmd/include` directory. However, if this function is used to create a navbar for a Github user web page, the html file should be stored in a sensible location inside the user pages repository, e.g., `leonawicz.github.io/assets`.

The common navigation bar html is included at the beginning of the body of the html for each web page in the project's website. `menu` is a vector of names for each dropdown menu. `submenus` is a list of vectors of menu options corresponding to each menu. `files` is a similar list of vectors. Each element is either an html file for a web page to be associated with the submenu link, "header" to indicate the corresponding name in `submenus` is only a group label and not a link to a web page, or "divider" to indicate placement of a bar for separating groups in a dropdown menu.

`theme` can be either `united` (default) or `cyborg`. Both are from Bootswatch. The function must apply some internal differentiation in the construction of the html navigation bar between themes. This is currently the only `rpm` function which attempts to handle multiple Bootswatch themes with different CSS tags.

```
# Functions for Github project websites
genNavbar <- function(htmlfile = "navbar.html", title, menu, submenus, files,
    title.url = "index.html", home.url = "index.html", site.url = "", site.name = "Github",
    theme = "united", include.home = FALSE) {
```

```
    if (!(theme %in% c("united", "cyborg")))
        stop("Only the following themes supported: united, cyborg.")

    navClassStrings <- function(x) {
        switch(x, united = c("brand", "nav-collapse collapse", "nav", "nav pull-right",
            "navbar-inner", "container", "", "btn btn-navbar", ".nav-collapse",
            "</div>\n"), cyborg = c("navbar-brand", "navbar-collapse collapse navbar-responsive-collapse
            "nav navbar-nav", "nav navbar-nav navbar-right", "container", "navbar-header",
            "      </div>\n", "navbar-toggle", ".nav-collapse", ""))
    }

    ncs <- navClassStrings(theme)

    fillSubmenu <- function(x, name, file, theme) {
        if (theme == "united")
            dd.menu.header <- "nav-header" else if (theme == "cyborg")
            dd.menu.header <- "dropdown-header"
        if (file[x] == "divider")
            return("              <li class=\"divider\"></li>\n")
        if (file[x] == "header")
            return(paste0("              <li class=\"", dd.menu.header, "\">",
                name[x], "</li>\n"))
        paste0("              <li><a href=\"", file[x], "\">", name[x], "</a></li>\n")
    }

    fillMenu <- function(x, menu, submenus, files, theme) {
        m <- menu[x]
        gs.menu <- gsub(" ", "-", tolower(m))
        s <- submenus[[x]]
        f <- files[[x]]
        if (s[1] == "empty") {
            y <- paste0("<li><a href=\"", f, "\">", m, "</a></li>\n")
        } else {
            y <- paste0("<li class=\"dropdown\">\n            <a href=\"", gs.menu,
                "\" class=\"dropdown-toggle\" data-toggle=\"dropdown\">", m,
                " <b class=\"caret\"></b></a>\n            <ul class=\"dropdown-menu\">\n",
                paste(sapply(1:length(s), fillSubmenu, name = s, file = f, theme = theme),
                  sep = "", collapse = ""), "            </ul>\n", collapse = "")
        }
    }

    if (include.home)
        home <- paste0("<li><a href=\"", home.url, "\">Home</a></li>\n          ") else home <- ""
    x <- paste0("<div class=\"navbar navbar-default navbar-fixed-top\">\n  <div class=\"",
        ncs[5], "\">\n    <div class=\"", ncs[6], "\">\n      <button type=\"button\" class=\"",
        ncs[8], "\" data-toggle=\"collapse\" data-target=\"", ncs[9], "\">\n        <span class=\"icon-b
        ncs[1], "\" href=\"", title.url, "\">", title, "</a>\n", ncs[7], "      <div class=\"",
        ncs[2], "\">\n        <ul class=\"", ncs[3], "\">\n          ", home,
        paste(sapply(1:length(menu), fillMenu, menu = menu, submenus = submenus,
            files = files, theme = theme), sep = "", collapse = "\n          "),
        "        </ul>\n        <ul class=\"", ncs[4], "\">\n          <a class=\"btn btn-primary\" href
        site.url, "\">\n            <i class=\"fa fa-github fa-lg\"></i>\n            ",
        site.name, "\n          </a>\n        </ul>\n      </div><!--/.nav-collapse -->\n    </div>\n  "
```

```
        ncs[10], "</div>\n", collpase = "")
    sink(htmlfile)
    cat(x)
    sink()
    x
}
```

3.2.13 genOutyaml

`genOutyaml` generates the `_out.yaml` file for yaml front-matter common to all html files in the project website. The file should be written to the project's `docs/Rmd` directory. `lib` specifies the library directory for any associated files. yaml `includes` for external html common to all project web pages in the site can also be specified with `header`, `before_body`, and `after_body`. These can be specified by file basename only (no path) and the function assumes these files are in the `docs/Rmd/include` directory. At this time all external libraries must be provided by the user, for example in `docs/Rmd/libs`. It is recommended. See the project repo [gh-pages](https://github.com/leonawicz/ProjectManagement/tree/gh-pages "gh-pages") branch for an example.

```
genOutyaml <- function(file, theme = "cosmo", highlight = "zenburn", lib = NULL,
    header = NULL, before_body = NULL, after_body = NULL) {
    output.yaml <- paste0("html_document:\n  self_contained: false\n  theme: ",
        theme, "\n  highlight: ", highlight, "\n  mathjax: null\n  toc_depth: 2\n")
    if (!is.null(lib))
        output.yaml <- paste0(output.yaml, "  lib_dir: ", lib, "\n")
    output.yaml <- paste0(output.yaml, "  includes:\n")
    if (!is.null(header))
        output.yaml <- paste0(output.yaml, "    in_header: ", header, "\n")
    if (!is.null(before_body))
        output.yaml <- paste0(output.yaml, "    before_body: ", before_body,
            "\n")
    if (!is.null(after_body))
        output.yaml <- paste0(output.yaml, "    after_body: ", after_body, "\n")
    sink(file)
    cat(output.yaml)
    sink()
    output.yaml
}
```

3.2.14 genAppDiv

`genAppDiv` generates an html file storing a container div element which organizes Shiny web applications. The function scans a directory of Shiny app subdirectories. This apps directory should be a local repository.

Specifically, `genAppDiv` looks for a named directory of image files. There should be one image per app, named exactly as the respective app directory is named. Only apps with corresponding images are built into the html container. If you wish to leave out, say, a developmental app from being linked to on you Github user website, do not include an image file for that app.

The container element includes an image link to each app's url as well as a link to the source code on Github. Although the function scans for images in directory inside a local repository, the images referenced in the output html are of course not local. They point to the same images stored on Github, hence why it is useful for the local directory of apps to be a Github repository. As an example, a repository may contain the directories, `app1`, `app2`, `app3`, and `images`.

This function will probably be removed in favor of the more general `genPanelDiv` function.

```
# Functions for Github user website
genAppDiv <- function(file = "C:/github/leonawicz.github.io/assets/apps_container.html",
    type = "apps", main = "Shiny Apps", apps.url = "http://shiny.snap.uaf.edu",
    github.url = "https://github.com/ua-snap/shiny-apps/tree/master", apps.dir = "C:/github/shiny-apps",
    img.loc = "_images/cropped", ...) {

    apps.img <- list.files(file.path(apps.dir, img.loc))
    apps <- sapply(strsplit(apps.img, "\\."), "[[", 1)
    x <- paste0("<div class=\"container\">\n  <div class=\"row\">\n    <div class=\"col-lg-12\">\n
        type, "\">", main, "</h3>\n      </div>\n    </div>\n  </div>\n  ")

    fillRow <- function(i, ...) {
        app <- apps[i]
        app.url <- file.path(apps.url, app)
        dots <- list(...)
        if (is.null(dots$col))
            col <- "warning" else col <- dots$col
        if (is.null(dots$panel.main))
            panel.main <- gsub("_", " ", app) else panel.main <- dots$panel.main
        if (length(panel.main) > 1)
            panel.main <- panel.main[i]
        x <- paste0("<div class=\"col-lg-4\">\n\t\t  <div class=\"bs-component\">\n\t\t\t<div class=\"pa
            col, "\">\n\t\t\t  <div class=\"panel-heading\"><h3 class=\"panel-title\">",
            panel.main, "</h3>\n\t\t\t  </div>\n\t\t\t  <div class=\"panel-body\"><a href=\"",
            app.url, "\" target=\"_blank\"><img src=\"", file.path(gsub("/tree/",
                "/raw/", github.url), img.loc, apps.img[i]), "\" alt=\"", apps[i],
            "\" width=100% height=200px></a><p></p>\n\t\t\t\t<div class=\"btn-group btn-group-justified\
            app.url, "\" target=\"_blank\" class=\"btn btn-success\">Launch</a>\n\t\t\t\t  <a href=\"",
            file.path(github.url, app), "\" target=\"_blank\" class=\"btn btn-info\">Github</a>\n\t\t\t\
    }

    n <- length(apps)
    seq1 <- seq(1, n, by = 3)
    y <- c()
    for (j in 1:length(seq1)) {
        ind <- seq1[j]:(seq1[j] + 2)
        ind <- ind[ind %in% 1:n]
        y <- c(y, paste0("<div class=\"row\">\n", paste0(sapply(ind, fillRow,
            ...), collapse = "\n"), "</div>\n"))
    }
    z <- "</div>\n"
    sink(file)
    sapply(c(x, y, z), cat)
    sink()
    cat("div container html created for Shiny Apps.\n")
}
```

3.2.15 genPanelDiv

`genPanelDiv` generates an html file storing a container div element which in its current state of development organizes two types of content: **R** projects and Shiny web applications.

The `type` argument can be one of `projects`, `apps`, `datavis`, or `gallery`. The purpose of the function is to generate an html file defining a container div element to display and reference either my **R** projects, my

Shiny apps, or my example visualization galleries.

Projects

For projects, the function scans a directory of local repositories and takes any directories found to be the names of projects. There is an `exclude` argument for dropping any known directories that are to be avoided. My defaults are `exclude="leonawicz.github.io", "shiny-apps"` since the first is just a local repository for my Github user account web site and not a "project" in the same sense of my other projects and the second is the local repository which is scanned by `genPanelDiv` when `type="apps"`.

Apps

For apps, the function scans a directory of Shiny app subdirectories. Unlike for projects, where `genPanelDiv` scans a directory of multiple local repositories, this apps directory should be a specific local repository. The apps contained within are not individual repositories. I have taken this approach for now simply because this is how my apps tend to be stored.

Specifically, the `genAppDiv` looks for a named directory of image files. There should be one image per app, named exactly as the respective app directory is named. Only apps with corresponding images are built into the html container. If you wish to leave out, say, a developmental app from being linked to on you Github user website, do not include an image file for that app.

The container element includes an image link to each app's url as well as a link to the source code on Github. Although the app scans for images in a local repository, the images referenced in the output html are of course not local. They point to the same images stored on Github, hence why it is useful for the local directory of apps to be a Github repository.

DataVis and Galleries

Whereas the first three types generate containers for the main Github user web page, I use `type="gallery"` to make a separate container html file of graphics for each panel occurring in my `datavis` container. These containers tend to be added to unique web pages. `datavis` is for highlighting a number of galleries whereas `gallery` is for the galleries' respective contents.

In order to use `type="datavis"` there must be a data visualization local repository. Mine is named `DataVisualizationExamples`, evident from the hardcoding currently in place within this function. Similar to when `type="apps"`, this repository includes a directory of images, in t his case one image for each gallery. Each image in this directory is named such that it identically matches another the name of a gallery images directory containing multiple images. As with `type="apps"`, gallery directories are only included if a corresponding thumbnail image in the images directory exists.

When `type="gallery"`, the behavior of `genPanelDiv` is most unique. For each gallery which exists, the function will make a unique html file with a gallery container element.

This function makes the more specific `genAppDiv` redundant and will likely replace it.

```
genPanelDiv <- function(outDir, type = "projects", main = "Projects", github.user = "leonawicz",
    prjs.dir = "C:/github", exclude = c("leonawicz.github.io", "shiny-apps",
        "DataVisExamples", ".git", "_images"), img.loc = "_images/small", lightbox = FALSE,
    include.buttons = TRUE, include.titles = TRUE, ...) {

    stopifnot(github.user %in% c("leonawicz", "ua-snap"))
    if (type == "apps") {
        filename <- "apps_container.html"
        web.url <- "http://shiny.snap.uaf.edu"
        gh.url.tail <- "shiny-apps/tree/master"
        atts <- " target=\"_blank\""
        go.label <- "Launch"
        prjs.dir <- file.path(prjs.dir, "shiny-apps")
        prjs.img <- list.files(file.path(prjs.dir, img.loc))
        prjs <- sapply(strsplit(prjs.img, "\\."), "[[", 1)
    }
    if (type == "projects") {
        filename <- "projects_container.html"
        web.url <- paste0("http://", github.user, ".github.io")
```

```
        gh.url.tail <- ""
        atts <- ""
        go.label <- "Website"
        prjs <- list.dirs(prjs.dir, full = TRUE, recursive = FALSE)
        prjs <- prjs[!(basename(prjs) %in% exclude)]
        prjs.img <- sapply(1:length(prjs), function(i, a) list.files(file.path(a[i],
            "plots"), pattern = paste0("^_", basename(a)[i])), a = prjs)
        prjs <- basename(prjs)
    }
    if (type == "datavis") {
        filename <- "data-visualizations_container.html"
        web.url <- paste0("http://", github.user, ".github.io")
        gh.url.tail <- "DataVisExamples/tree/master"
        atts <- ""
        go.label <- "See More"
        prjs.dir <- file.path(prjs.dir, "DataVisExamples")
        prjs.img <- list.files(file.path(prjs.dir, img.loc))
        prjs <- sapply(strsplit(prjs.img, "\\."), "[[", 1)
    }
    if (type == "gallery") {
        web.url <- paste0("http://", github.user, ".github.io")
        gh.url.tail <- "DataVisExamples/tree/master"
        if (lightbox)
            atts1 <- " data-lightbox=\"ID\"" else atts1 <- ""
        go.label <- "Expand"
        prjs <- list.dirs(file.path(prjs.dir, "DataVisExamples"), full = T,
            recursive = F)
        prjs <- prjs[!(basename(prjs) %in% exclude)]
        prjs.img <- lapply(1:length(prjs), function(x, files, imgDir) list.files(path = file.path(files
            imgDir), recursive = FALSE), files = prjs, imgDir = img.loc)
        prjs <- basename(prjs)
        filename <- tolower(paste0("gallery-", gsub(" ", "-", gsub(" - ", " ",
            prjs)), ".html"))
    }
    gh.url <- file.path("https://github.com", github.user, gh.url.tail)

    fillRow <- function(i, ...) {
        prj <- panels[i]
        dots <- list(...)
        if (is.null(dots$col))
            col <- "warning" else col <- dots$col
        if (is.null(dots$panel.main))
            panel.main <- gsub(" - ", ": ", gsub("_", " ", prj)) else panel.main <- dots$panel.main
        if (length(panel.main) > 1)
            panel.main <- panel.main[i]
        if (type == "apps")
            img.src <- file.path(gsub("/tree/", "/raw/", gh.url), img.loc, prjs.img[i])
        if (type == "projects")
            img.src <- file.path(gh.url, prj, "raw/master/plots", prjs.img[i])
        if (type == "datavis")
            img.src <- file.path(gsub("/tree/", "/raw/", gh.url), img.loc, prjs.img[i])
        if (type != "gallery") {
            if (type == "datavis")
                pfx <- "gallery-" else pfx <- ""
```

```
            web.url <- file.path(web.url, tolower(paste0(pfx, gsub("_", "-",
                gsub("_-_", "-", prj)), ".html")))
        } else {
            prj <- prjs[p]
            img.src <- file.path(gsub("/tree/", "/raw/", gh.url), prjs[p], img.loc,
                panels[i])
            web.url <- file.path(gsub("/tree/", "/raw/", gh.url), prjs[p], panels[i])
            if (lightbox)
                atts <- gsub("ID", gsub(" - ", ": ", gsub("_", " ", prjs[p])),
                  atts1) else atts <- atts1
        }
        if (include.titles) {
            panel.title <- paste0("<div class=\"panel-heading\"><h3 class=\"panel-title\">",
                panel.main, "</h3>\n          </div>\n          ")
        } else panel.title <- ""
        if (include.buttons) {
            panel.buttons <- paste0("<div class=\"btn-group btn-group-justified\">\n\t\t\t<a href=\"",
                web.url, "\"", atts, " class=\"btn btn-success\">", go.label,
                "</a>\n\t\t\t<a href=\"", file.path(gh.url, prj), "\" class=\"btn btn-info\">Github</a>\
        } else panel.buttons <- ""
        x <- paste0("    <div class=\"col-lg-4\">\n      <div class=\"bs-component\">\n        <div clas
            col, "\">\n          ", panel.title, "<div class=\"panel-body\"><a href=\"",
            web.url, "\"", atts, "><img src=\"", img.src, "\" alt=\"", panel.main,
            "\" width=100% height=200px></a><p></p>\n          ", panel.buttons,
            "  </div>\n        </div>\n      </div>\n    </div>\n  ")
    }

    for (p in 1:length(filename)) {
        if (type == "gallery") {
            panels <- prjs.img[[p]]
            main <- gsub(" - ", ": ", gsub("_", " ", prjs[p]))
        } else panels <- prjs
        n <- length(panels)
        seq1 <- seq(1, n, by = 3)
        x <- paste0("<div class=\"container\">\n  <div class=\"row\">\n    <div class=\"col-lg-12\">\n
            type, "\">", main, "</h3>\n      </div>\n    </div>\n  </div>\n  ")
        y <- c()
        for (j in 1:length(seq1)) {
            ind <- seq1[j]:(seq1[j] + 2)
            ind <- ind[ind %in% 1:n]
            y <- c(y, paste0("<div class=\"row\">\n", paste0(sapply(ind, fillRow,
                ...), collapse = "\n"), "</div>\n"))
        }
        z <- "</div>\n"
        sink(file.path(outDir, filename[p]))
        sapply(c(x, y, z), cat)
        sink()
        cat("div container html file created.\n")
    }
}
```

3.2.16 htmlHead

`htmlHead` is useful for including javascript and CSS stylesheets in the head of an html document. Stylesheet arguments can be passed along as well in proper order.

```
htmlHead <- function(author = "Matthew Leonawicz", title = author, script.paths = NULL,
    stylesheet.paths, stylesheet.args = vector("list", length(path.stylesheets)),
    ...) {
    x <- paste0("<!DOCTYPE html>\n\n<html xmlns=\"http://www.w3.org/1999/xhtml\">\n\n<head>\n\n<meta cha
        author, " />\n\n<title>", title, "</title>\n")

    if (is.character(script.paths))
        x <- c(x, paste0(paste0("<script src=\"", script.paths, "\"></script>",
            collapse = "\n"), "\n"))

    x <- c(x, "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\" />\n")

    if (is.character(stylesheet.paths)) {
        n <- length(stylesheet.paths)
        stopifnot(is.list(stylesheet.args))
        stopifnot(length(stylesheet.args) == n)
        for (i in 1:n) {
            string <- ""
            if (is.list(stylesheet.args[i])) {
                v <- stylesheet.args[i]
                arg <- names(v)
                if (is.character(arg) && all(arg != ""))
                  string <- paste0(" ", paste(arg, paste0("\"", v, "\""), sep = "=",
                    collapse = " "))
            }
            x <- c(x, paste0("<link rel=\"stylesheet\" href=\"", stylesheet.paths[i],
                "\"", string, ">\n"))
        }
    }

    x <- c(x, "</head>\n")
    x

}
```

3.2.17 htmlBodyTop

`htmlBodyTop` currently is used for including custom CSS and a background image in the html body. CSS can be included as a text string or as a path to a CSS file.

```
htmlBodyTop <- function(css.file = NULL, css.string = NULL, background.image = "",
    include.default = TRUE, ...) {
    x <- "<style type = \"text/css\">\n"

    default <- paste0("\n\t.main-container {\n\t  max-width: 940px;\n\t  margin-left: auto;\n\t  margin-
        background.image, "\");\n\t  background-attachment: fixed;\n\t  background-size: 1920px 1080px;\

    if (!is.null(css.file))
        y <- readLines(css.file) else y <- ""
```

```
    if (!is.null(css.string))
        y <- c(y, css.string)
    if (include.default)
        y <- c(default, y)

    z <- "\n</style>\n\t<div class=\"container-fluid main-container\">\n\t"

    c(x, y, z)
}
```

3.2.18 htmlBottom

`htmlBottom` does not do anything else at this time other than close up the html document.

```
htmlBottom <- function(...) {
    # temporary
    "</body>\n\t</html>"
}
```

3.2.19 genUserPage

`genUserPage` generates a Github user account web page by combining precompiled html files of container elements made using `genPanelDiv` as well as various lingering hardcoded elements for my own work. I use this function to produce my main Github user page, the `index.html`, as well as supplemental gallery pages.

```
genUserPage <- function(file = "C:/github/leonawicz.github.io/index.html", containers = NULL,
    navbar = "", ...) {
    x1 <- htmlHead(...)
    x2 <- htmlBodyTop(...)
    if (!is.null(containers))
        x3 <- sapply(containers, function(x) paste0(paste0(readLines(x), collapse = "\n"),
            "\n\n")) else x3 <- ""
    x4 <- htmlBottom(...)
    nb <- if (file.exists(navbar) && substr(navbar, nchar(navbar) - 4, nchar(navbar)) ==
        ".html")
        nb <- paste0(paste0(readLines(navbar), collapse = "\n"), "\n\n")
    sink(file)
    sapply(c(x1, x2, nb, x3, x4), cat)
    sink()
    cat("Github User page html file created.\n")
}
```