Regular readers will recall the “utility belt”. This is a follow-up to a request made asking for a list of all the % infix functions in those files.

A Brief Introduction to utility belts

Many R package authors (including myself) lump a collection of small, useful functions into some type of utils.R file and usually do not export the functions since they are (generally) designed to work on package internals rather than expose their functionality via the exported package API. Just like Batman’s utility belt, which can be customized for any mission, any set of utilities in a given R package will also likely be different from those in other packages.

I thought it would be neat to take a look at:

* just how many packages have one or more util\*.R files and what the most common file names are for them;
* utility function naming preferences — specifically snake-case, camel-case or dot-case
* what the most common “utility” functions names are across the packages
* coding style — specifically compare ratios of white space, full-line comments to code size

I made sure to show a few examples of where a better search pattern would have helped ensure lines like the three at the bottom of that listings aren’t included. But, we all often have to deal with imperfect data, so we’ll make sure to deal with that during the ingestion & cleanup process.

library(stringi)

library(hrbrthemes)

library(archive)

library(tidyverse)

# I ran readr::type\_convert() once and it returns this column type spec. By using it

# for subsequent conversions, we'll gain reproducibility and data format change

# detection capabilities "for free"

cols(

permsissions = col\_character(),

links = col\_integer(),

owner = col\_character(),

group = col\_character(),

size = col\_integer(),

month = col\_character(),

day = col\_integer(),

year\_hr = col\_character(),

path = col\_character()

) -> tar\_cols

# Now, we parse the tar verbose ('ls -l') listing

stri\_read\_lines("~/Data/pkutils.txt") %>% # stringi was loaded so might as well use it

stri\_split\_regex(" +", 9, simplify = TRUE) %>% # split input into 9 columns

as\_data\_frame() %>% # ^^ returns a matrix but data frames are more useful for our work

set\_names(names(tar\_cols$cols)) %>% # column names are useful and we can use our colspec for it

type\_convert(col\_types = tar\_cols) %>% # see comment block before cols()

mutate(day = sprintf("%02d", day)) %>% # now we'll work on getting the date pieces to be a Date

mutate(year\_hr = case\_when( # the year\_hr field can be either %Y or %H:%M depending on file 'recency'

stri\_detect\_fixed(year\_hr, ":") &

(month %in% c("Jan", "Feb", "Mar", "Apr")) ~ "2018", # if %H:%M but 'starter' months it's 2018

stri\_detect\_fixed(year\_hr, ":") &

(month %in% c("Dec", "Nov", "Oct", "Sep", "Aug", "Jul", "Jun")) ~ "2017", # %H:%M & 'end' months

TRUE ~ year\_hr # already in %Y format

)) %>%

mutate(date= lubridate::mdy(sprintf("%s %s, %s", month, day, year\_hr))) %>% # get a Date

mutate(pkg = stri\_match\_first\_regex(path, "^(.\*)/R/")[,2]) %>% # extract package name (stri\_extract is also usable here)

mutate(fil = basename(path)) %>% # extrafct just the file name

filter(!is.na(pkg)) %>% # handle one type of wrongly included file

filter(!stri\_detect\_fixed(pkg, "/")) %>% # ande another

filter(!is.na(path)) -> xdf # and another; but we're done so we close with an assignment

glimpse(xdf)

## Observations: 1,746

## Variables: 12

## $ permsissions <chr> "-rw-r--r--", "-rw-r--r--", "-rw-r--r--", "-rw-r-...

## $ links <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...

## $ owner <chr> "hornik", "ligges", "hornik", "ligges", "ligges",...

## $ group <chr> "users", "users", "users", "users", "users", "her...

## $ size <int> 1658, 12609, 0, 4127, 121, 52, 36977, 34198, 3676...

## $ month <chr> "Jun", "Dec", "Feb", "Aug", "Jan", "Aug", "Jan", ...

## $ day <chr> "05", "13", "24", "30", "19", "10", "06", "10", "...

## $ year\_hr <chr> "2016", "2016", "2017", "2017", "2017", "2017", "...

## $ path <chr> "AHR/R/util.R", "ALA4R/R/utilities\_internal.R", "...

## $ date <date> 2016-06-05, 2016-12-13, 2017-02-24, 2017-08-30, ...

## $ pkg <chr> "AHR", "ALA4R", "AWR.Kinesis", "AlphaVantageClien...

## $ fil <chr> "util.R", "utilities\_internal.R", "utils.R", "uti...

To the analysis!

Finding the Utility of ‘util’s

A careful look at the glimpse() listing shows we have 1,745 files that begin with util, but how many *packages* have at least one util files?

nrow(distinct(xdf, pkg))

## [1] 1397

That’s roughly 10% of CRAN, but doesn’t mean other packages do not have “utility belt” functions since other authors may have just been more creative or deliberate with their file naming conventions.

Readers with keen eyes may have noticed we spent some deliberate CPU cycles to get a Date column. Part of that was to show how to do that (mostly as an example for folks new to R) but we also did it to ask temporal questions, such as “Are package ‘utility belts’ a “new” thing?”. The data suggests that utility belts are products/attributes of more recently published or updated packages:

distinct(xdf, pkg, date) %>%

mutate(yr = as.integer(lubridate::year(date))) %>%

count(yr) %>%

complete(yr, fill=list(n=0)) %>%

ggplot(aes(yr, n)) +

geom\_col(fill="lightslategray", width=0.65) +

labs(

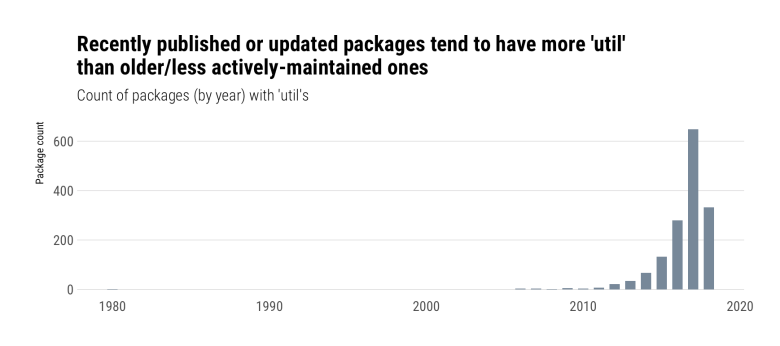
x = NULL, y = "Package count",

title = "Recently published or updated packages tend to have more 'util'\nthan older/less actively-maintained ones",

subtitle = "Count of packages (by year) with 'util's"

) +

theme\_ipsum\_rc(grid="Y")



We *could* answer this more completely by going through the CRAN archives for all these packages, but for now we’ll just see which packages might have helped set this trend going:

distinct(xdf, pkg, date) %>%

arrange(date) %>%

print(n=20)

## # A tibble: 1,540 x 2

## date pkg

## 1 1980-01-01 bsts

## 2 2006-06-28 evdbayes

## 3 2006-11-29 hexView

## 4 2006-12-17 StatDataML

## 5 2007-10-05 tpr

## 6 2007-11-07 seqinr

## 7 2007-11-26 registry

## 8 2008-07-25 ramps

## 9 2008-10-23 RobAStBase

## 10 2009-02-23 vcd

## 11 2009-06-26 ttutils

## 12 2009-07-03 histogram

## 13 2009-11-27 polynom

## 14 2009-11-27 tau

## 15 2010-01-05 itertools

## 16 2010-01-22 tableplot

## 17 2010-06-09 rbugs

## 18 2011-03-17 playwith

## 19 2011-05-11 marelac

## 20 2011-10-11 timeSeries

## # ... with 1,520 more rows

Going back to our corpus, what are the most common names for these utility belt files?

## count(xdf, fil, sort=TRUE) %>%

## mutate(pct = scales::percent(n/sum(n))) %>%

## print(n=20)

## # A tibble: 409 x 3

## fil n pct

## 1 utils.R 865 49.5%

## 2 utilities.R 145 8.3%

## 3 util.R 134 7.7%

## 4 utils.r 68 3.9%

## 5 utility.R 47 2.7%

## 6 utility\_functions.R 25 1.4%

## 7 util.r 16 0.9%

## 8 utilities.r 14 0.8%

## 9 utils-pipe.R 9 0.5%

## 10 utilityFunctions.R 6 0.3%

## 11 utils-format.r 3 0.2%

## 12 util\_functions.R 2 0.1%

## 13 util\_rescale.R 2 0.1%

## 14 util-aux.R 2 0.1%

## 15 util-checkparam.R 2 0.1%

## 16 util-startarg.R 2 0.1%

## 17 utilcmst.R 2 0.1%

## 18 utilhot.R 2 0.1%

## 19 utilities\_internal.R 2 0.1%

## 20 utility-functions.R 2 0.1%

## # ... with 389 more rows

Over 50% of other CRAN packages are as “un-creative” as I am when it comes to naming these files.

Let’s see how packed these belts are:

ggplot(xdf, aes(x="", size)) +

ggbeeswarm::geom\_quasirandom(

fill="lightslategray", color="white",

alpha=1/2, stroke=0.25, size=3, shape=21

) +

geom\_boxplot(fill="#00000000", outlier.colour = "#00000000") +

geom\_text(

data=data\_frame(), aes(x=-Inf, y=median(xdf$size), label="Median:\n2,717"),

hjust = 0, family = font\_rc, size = 3, color = "lightslateblue"

) +

scale\_y\_comma(

name = "File size", trans="log10", limits=c(NA, 200000),

breaks = c(10, 100, 1000, 10000, 100000)

) +

labs(

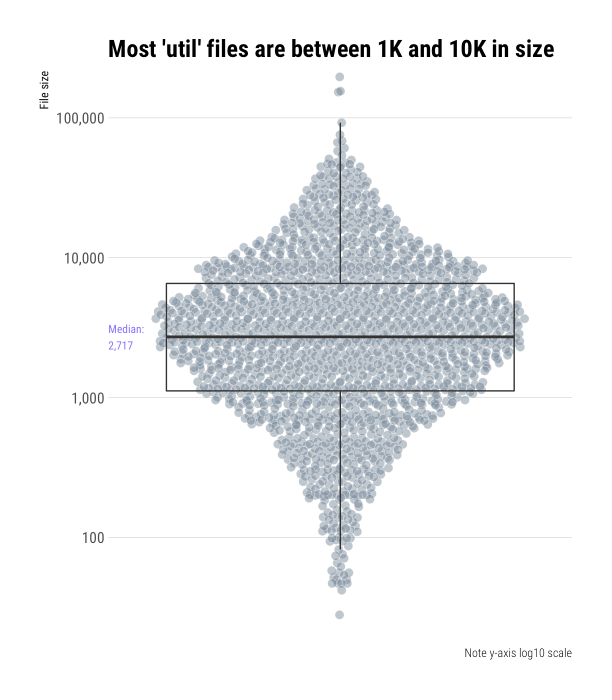
x = NULL,

title = "Most 'util' files are between 1K and 10K in size",

caption = "Note y-axis log10 scale"

) +

theme\_ipsum\_rc(grid="Y")



We’ll need to do a bit more data collection to answer the last two questions.

Focus on Functions

To examine function names and source code statistics, we’ll need to read in the contents of each file and parse them. Let’s do that first bit with some help from the [archive](https://github.com/jimhester/archive) package which will help us open up these compressed tar files and pull out the file(s) we need from them vs have to code this up more manually.

Again, this code is only reproducible if you have CRAN handy, but soon (*promise!*) you’ll have a file you can work with for the remainder of the post:

extract\_source <- function(pkg, fil, .pb = NULL) {

if (!is.null(.pb)) .pb$tick()$print()

list.files(

path = "/cran/src/contrib", # my path to local CRAN

pattern = sprintf("^%s\_.\*gz", pkg), # rough pattern for the package archive filename

recursive = FALSE,

full.names = TRUE

) -> tgt

con <- archive\_read(tgt[1], fil)

src <- readLines(con, warn = FALSE)

close(con)

paste0(src, collapse="\n")

}

pb <- progress\_estimated(nrow(xdf))

xdf <- mutate(xdf, file\_src = map2\_chr(pkg, path, extract\_source, .pb=pb))

That (on-drive) ~10MB data frame is in <https://rud.is/dl/utility-belt.rds>. The rest of the post builds off of it so you can start coding along at home now.

Let’s extract the function names:

# we'll use these two functions to help test whether bits

# of our parsed code are, indeed, functions.

#

# Alternately: "I heard you liked functions so I made

# functions to help you find functions"

#

# we could have used `rlang` helpers here, but I had these

# handy from pre-`rlang` days.

is\_assign <- function(x) {

as.character(x) %in% c('<-', '=', '<<-', 'assign')

}

is\_func <- function(x) {

is.call(x) &&

is\_assign(x[[1]]) &&

is.call(x[[3]]) &&

(x[[3]][[1]] == quote(`function`))

}

read\_rds("~/Data/utility-belt.rds") %>% # I have this file in ~/Data; change this for your location

mutate(parsed = map(file\_src, ~parse(text = .x, keep.source = TRUE))) %>% # parse each file

mutate(func\_names = map(parsed, ~{ # go through parsed file

keep(.x, is\_func) %>% # and only keep functions

map(~as.character(.x[[2]])) %>% # extract the function name

flatten\_chr() # return a character vector

})) -> xdf

With those handy, we can see if there are any commonalities across all these packages:

select(xdf, pkg, fil, func\_names) %>%

unnest() %>%

count(func\_names, sort=TRUE) %>%

print(n=20)

## func\_names n

## 1 %||% 84

## 2 compact 19

## 3 isFALSE 19

## 4 assertthat::on\_failure 16

## 5 is\_windows 16

## 6 trim 14

## 7 .on Load 13 # (IRL there's no space here but the WP input sanitizer hates this word due to js abuse

## 8 names2 12

## 9 dots 11

## 10 is\_string 11

## 11 vlapply 11

## 12 .onAttach 10

## 13 error.bars 10

## 14 normalize 10

## 15 vcapply 10

## 16 cat0 9

## 17 collapse 9

## 18 err 9

## 19 getmin 9

## 20 is\_dir 9

## # ... with 1.252e+04 more rows

We can also see if there are common case conventions:

select(xdf, pkg, fil, func\_names) %>%

unnest() %>%

mutate(is\_camel = (!stri\_detect\_fixed(func\_names, "\_")) &

(!stri\_detect\_regex(func\_names, "[[:alpha:]]\\.[[:alpha:]]")) &

(stri\_detect\_regex(func\_names, "[A-Z]"))) %>%

mutate(is\_dotcase = stri\_detect\_regex(func\_names, "[[:alpha:]]\\.[[:alpha:]]")) %>%

mutate(is\_snake = stri\_detect\_fixed(func\_names, "\_") &

(!stri\_detect\_regex(func\_names, "[[:alpha:]]\\.[[:alpha:]]"))) -> case\_hunt

count(case\_hunt, is\_camel, is\_dotcase, is\_snake) %>%

mutate(pct = scales::percent(n/sum(n))) %>%

mutate(description = c(

"one-'word' names",

"snake\_case",

"dot.case",

"camelCase"

)) %>%

arrange(n) %>%

mutate(description = factor(description, description)) %>%

ggplot(aes(description, n)) +

geom\_col(fill="lightslategray", width=0.65) +

geom\_label(aes(y = n, label=pct), label.size=0, family=font\_rc, nudge\_y=150) +

scale\_y\_comma("Number of functions") +

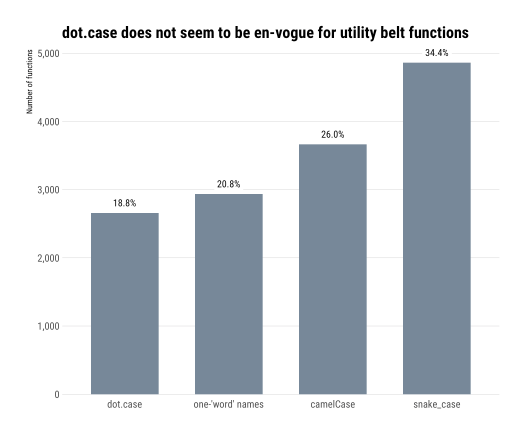
labs(

x=NULL,

title = "dot.case does not seem to be en-vogue for utility belt functions"

) +

theme\_ipsum\_rc(grid="Y")



I had a hunch that isX…()/is\_x…() could be likely names for utility belt functions, so let’s normalize the function names to snake\_case and see if that’s true:

select(xdf, pkg, fil, func\_names) %>%

unnest() %>%

filter(stri\_detect\_regex(func\_names, "^(\\.is|is)")) %>%

mutate(func\_names = snakecase::to\_snake\_case(func\_names)) %>%

count(func\_names, sort=TRUE)

## # A tibble: 547 x 2

## func\_names n

## 1 is\_false 24

## 2 is\_windows 19

## 3 is\_string 18

## 4 is\_empty 13

## 5 is\_dir 11

## 6 is\_formula 11

## 7 is\_installed 11

## 8 is\_linux 9

## 9 is\_na 9

## 10 is\_error 8

## # ... with 537 more rows

Only 5% (819) out of 14,123 extracted function names are is\_; not overwhelming, but a respectable slice.

There are more questions we could ask of function names and styles, but we’ll leave some work for y’all to do on your own.

Let’s head over to the final ~~rooftop~~ exercise.

Code, Comment & Blank Line Density

Since we have the raw source, we can also take a look at coding style. There are many questions we could ask here and more than a few packages we could draw on to help answer them. For now, we’ll just take a look at the mean ratios of comments and blank lines to code across the packages in this utility belt corpus and give you the opportunity to tease out other interesting tidbits such as “what base R and other package functions are most often used in utility belt functions?” or “are package authors using evil = for assignment or proper <-?”.

xdf %>%

mutate(

num\_lines = stri\_count\_fixed(xdf$file\_src, "\n"),

num\_blank\_lines = stri\_count\_regex(xdf$file\_src, "^[[:space:]]\*$", opts\_regex = stri\_opts\_regex(multiline=TRUE)),

num\_whole\_line\_comments = lengths(cmnt\_df$comments),

comment\_density = num\_whole\_line\_comments / (num\_lines - num\_blank\_lines - num\_whole\_line\_comments),

blank\_density = num\_blank\_lines / (num\_lines - num\_whole\_line\_comments)

) %>%

select(-permsissions, -links, -owner, -group, month, -day, -year\_hr) -> xdf

# now compute mean ratios

group\_by(xdf, pkg) %>%

summarise(

`Comment-to-code Ratio` = mean(comment\_density),

`Blank lines-to-code Ratio` = mean(blank\_density)

) %>%

ungroup() %>%

filter(!is.infinite(`Comment-to-code Ratio`)) %>%

filter(!is.nan(`Comment-to-code Ratio`)) %>%

filter(!is.infinite(`Blank lines-to-code Ratio`)) %>%

filter(!is.nan(`Blank lines-to-code Ratio`)) %>%

gather(measure, value, -pkg) -> code\_ratios

# we want to label the median values

group\_by(code\_ratios, measure) %>%

summarise(median = median(value)) -> code\_ratio\_meds

ggplot(code\_ratios, aes(measure, value, group=measure)) +

ggbeeswarm::geom\_quasirandom(

fill="lightslategray", color="#2b2b2b", alpha=1/2,

stroke=0.25, size=3, shape=21

) +

geom\_boxplot(fill="#00000000", outlier.colour = "#00000000") +

geom\_label(

data = code\_ratio\_meds,

aes(-Inf, c(0.3, 5), label=sprintf("Median:\n%s", round(median, 2)), group=measure),

family = font\_rc, size=3, color="lightslateblue", hjust = 0, label.size=0

) +

scale\_y\_continuous() +

labs(

x = NULL, y = NULL,

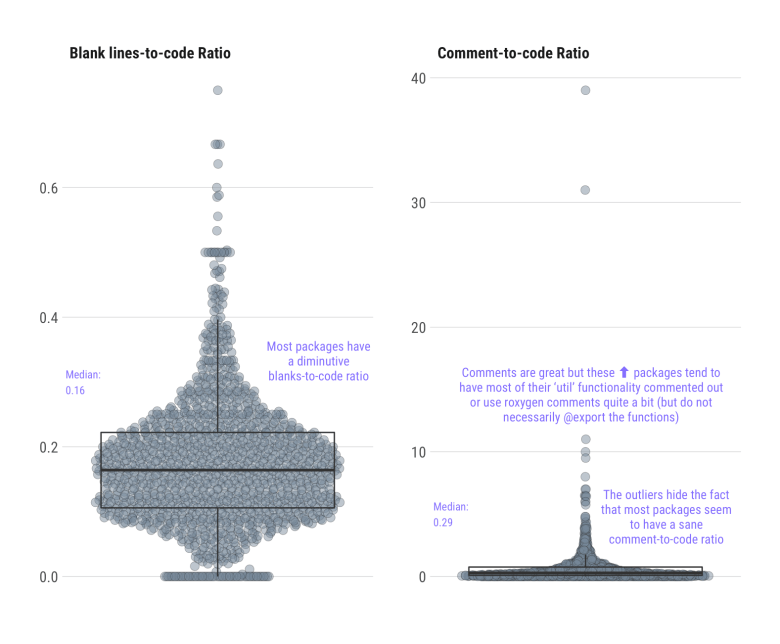
caption = "Note free y scale"

) +

facet\_wrap(~measure, scales="free") +

theme\_ipsum\_rc(grid="Y", strip\_text\_face = "bold") +

theme(axis.text.x=element\_blank())



We’re going to:

* collect up all of the sources
* parse them
* find all the definitions of % infix functions
* write them to a file

We’ll start by grabbing the data as attached in this repository and look at it as a refresher:

library(stringi)

library(tidyverse)

utils <- read\_rds(utility-belt.rds")

utils

## # A tibble: 1,746 x 13

## permsissions links owner group size month day year\_hr path date pkg fil file\_src

## 1 -rw-r--r-- 0 hornik users 1658 Jun 05 2016 AHR/R… 2016-06-05 AHR util… "## \\int f(x)dg(x) …

## 2 -rw-r--r-- 0 ligges users 12609 Dec 13 2016 ALA4R… 2016-12-13 ALA4R util… "## some utility fun…

## 3 -rw-r--r-- 0 hornik users 0 Feb 24 2017 AWR.K… 2017-02-24 AWR.… util… ""

## 4 -rw-r--r-- 0 ligges users 4127 Aug 30 2017 Alpha… 2017-08-30 Alph… util… "#\n#' Assign API ke…

## 5 -rw-r--r-- 0 ligges users 121 Jan 19 2017 Amylo… 2017-01-19 Amyl… util… "make\_decision <- fu…

## 6 -rw-r--r-- 0 herbrandt herbrandt 52 Aug 10 2017 BANES… 2017-08-10 BANE… util… "#' @importFrom dply…

## 7 -rw-r--r-- 0 ripley users 36977 Jan 06 2015 BEQI2… 2015-01-06 BEQI2 util… "#' \tRemove Redunda…

## 8 -rw-r--r-- 0 hornik users 34198 May 10 2017 BGDat… 2017-05-10 BGDa… util… "# A more memory-eff…

## 9 -rwxr-xr-x 0 ligges users 3676 Aug 14 2016 BGLR/… 2016-08-14 BGLR util… "\n readBinMat=funct…

## 10 -rw-r--r-- 0 ripley users 2547 Feb 04 2015 BLCOP… 2015-02-04 BLCOP util… "###################…

## # ... with 1,736 more rows

Note that we somewhat expected the file source to potentially come in handy at a later date and also expected the need to revisit that post.

Now, let’s find all the source files with at least one infix definition, collect them together and parse them so we can do more code spelunking:

filter(utils, stri\_detect\_fixed(file\_src, "`%")) %>% # only find sources with infix definitions

pull(file\_src) %>%

paste0(collapse="\n\n") %>%

parse(text = ., keep.source=TRUE) -> infix\_src

str(infix\_src, 1)

## length 1364 expression(dplyr::`%>%`, `%||%` <- function(a, b) if (is.null(a)) b else a, get\_pkg\_path <- function(ctx) { pkg\_| \_\_truncated\_\_ ...

## - attr(\*, "srcref")=List of 1364

## - attr(\*, "srcfile")=Classes 'srcfilecopy', 'srcfile'

## - attr(\*, "wholeSrcref")= 'srcref' int [1:8] 1 0 15768 0 0 0 1 15768

## ..- attr(\*, "srcfile")=Classes 'srcfilecopy', 'srcfile'

We can now take all of that lovely parsed source and tokenize it to work with the discrete elements in a very tidy manner:

infix\_parsed <- tbl\_df(getParseData(infix\_src)) # tbl\_df() is mainly for pretty printing

infix\_parsed

## # A tibble: 118,242 x 9

## line1 col1 line2 col2 id parent token terminal text

## 1 1 1 1 24 1 -10 COMMENT TRUE #' @impor…

## 2 2 1 2 10 4 -10 COMMENT TRUE #' @export

## 3 3 1 3 12 10 0 expr FALSE ""

## 4 3 1 3 5 7 10 SYMBOL\_PACKAGE TRUE dplyr

## 5 3 6 3 7 8 10 NS\_GET TRUE ::

## 6 3 8 3 12 9 10 SYMBOL TRUE `%>%`

## 7 5 1 5 49 51 0 expr FALSE ""

## 8 5 1 5 6 16 18 SYMBOL TRUE `%||%`

## 9 5 1 5 6 18 51 expr FALSE ""

## 10 5 8 5 9 17 51 LEFT\_ASSIGN TRUE <-

## # ... with 118,232 more rows

We just need to find a sequence of tokens that make up a function definition, then whittle those down to ones that look like our % infix names:

pat <- c("SYMBOL", "expr", "LEFT\_ASSIGN", "expr", "FUNCTION") # pattern for function definition

# find all of ^^ sequences (there's a good twitter discussion on this abt a month ago)

idx <- which(infix\_parsed$token == pat[1]) # find location of match of start of seq

# look for the rest of the sequences starting at each idx position

map\_lgl(idx, ~{

all(infix\_parsed$token[.x:(.x+(length(pat)-1))] == pat)

}) -> found

f\_defs <- idx[found] # starting indices of all the places where functions are defined

# filter ^^ to only find infix ones

infix\_defs <- f\_defs[stri\_detect\_regex(infix\_parsed$text[f\_defs], "^`\\%")]

# there aren't too many, but remember we're just searching `util` functions

length(infix\_defs)

## [1] 106

Now, write it out to a file so we can peruse the infix functions:

# nuke a file and fill it with the function definition

cat("", sep="", file="infix\_functions.R")

walk2(

getParseText(infix\_parsed, infix\_parsed$id[infix\_defs]), # extract the infix name

getParseText(infix\_parsed, infix\_parsed$id[infix\_defs + 3]), # extract the function definition body

~{

cat(.x, " <- ", .y, "\n\n", sep="", file="infix\_functions.R", append=TRUE)

}

)

There are 106 of them so you can find the extracted ones below:

|  |
| --- |
| `%||%` <- function(a, b) if (is.null(a)) b else a |
|  |  |
|  | `%diag\*%` <- function(d, X) d \* X |
|  |  |
|  | `%\*diag%` <- function(X, d) t(t(X) \* d) |
|  |  |
|  | `%nin%` <- function(x, table) !(x %in% table) |
|  |  |
|  | `%sub\_in%` <- function(x, table) x[x %in% table] |
|  |  |
|  | `%sub\_nin%` <- function(x, table) x[x %nin% table] |
|  |  |
|  | `%notchin%` <- function(lhs, rhs) { |
|  | !{lhs %chin% rhs} |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) ifelse(is.null(x), y, x) |
|  |  |
|  | `%||%` <- function(x, y) if (is.null(x)) y else x |
|  |  |
|  | `%||%` <- function(x, y) if (is.null(x)) y else x |
|  |  |
|  | `%||%` <- function(x, default\_val) { |
|  | if (is.null(x)) return(default\_val) |
|  | x |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%p%` <- function(e1,e2) return(paste0(e1,e2)) |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (!is.null(x)) { |
|  | x |
|  | } else { |
|  | y |
|  | } |
|  | } |
|  |  |
|  | `%==%` <- function(x, y) identical(x, y) |
|  |  |
|  | `%!=%` <- function(x, y) !identical(x, y) |
|  |  |
|  | `%:::%` <- function(pkg, fun) { |
|  | getFromNamespace(fun, asNamespace(pkg)) |
|  | } |
|  |  |
|  | `%||%` <- function(left, right){ |
|  | if (!is.null(left)){ |
|  | return(left) |
|  | } |
|  | right |
|  | } |
|  |  |
|  | `%pin%` <- function(x, table) pmatch(x, table, nomatch = 0L) > 0L |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x) || length(x) <= 0) { |
|  | y |
|  | } else { |
|  | x |
|  | } |
|  | } |
|  |  |
|  | `%??%` <- function(lhs, rhs) |
|  | if (is\_empty(lhs)) rhs else lhs |
|  |  |
|  | `%In%` <- function(x,Intv) |
|  | { |
|  | if (is.integer(x)) x <- as.double(x) |
|  | if (is.matrix(Intv)) |
|  | { |
|  | Intv <- Intervals(Intv) |
|  | } |
|  | distance\_to\_nearest(x,Intv) == 0 |
|  | } |
|  |  |
|  | `%<<%` <- function(a, b) bitwShiftL(a, b) |
|  |  |
|  | `%>>%` <- function(a, b) bitwShiftR(a, b) |
|  |  |
|  | `%&%` <- function(a, b) bitwAnd(a, b) |
|  |  |
|  | `%notin%` <- function(x, table) match(x, table, nomatch = 0L) == 0L |
|  |  |
|  | `%or%` <- function(lhs, rhs) if (is.null(lhs)) rhs else lhs |
|  |  |
|  | `%||%` <- function (x, y) if (is.null(x)) y else x |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) return(y) |
|  | x |
|  | } |
|  |  |
|  | `%~~%` <- function(x, y) { |
|  | if (length(x) == 0L) return(y) |
|  | x |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) y else x |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) if(is.null(x)) y else x |
|  |  |
|  | `%||%` <- function(x, y) if (is.null(x)) y else x |
|  |  |
|  | `%||%` <- function(lhs, rhs) { |
|  | if (is.null(lhs)) { |
|  | rhs |
|  | } else { |
|  | lhs |
|  | } |
|  | } |
|  |  |
|  | `%R%` <- function(lhs, rhs){ |
|  | if(length(lhs)) lhs else rhs |
|  | } |
|  |  |
|  | `%M%` <- function(lhs, rhs) { |
|  | if (lhs < rhs) { |
|  | old.lhs <- lhs |
|  | lhs <- rhs |
|  | rhs <- old.lhs |
|  | } |
|  | x <- lhs %/% rhs |
|  | y <- lhs %% rhs |
|  | return(c(quotient = x, remainder = y)) |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) { |
|  | y |
|  | } else { |
|  | x |
|  | } |
|  | } |
|  |  |
|  | `%notin%` <- function(x, y) { |
|  | !(x %in% y) |
|  | } |
|  |  |
|  | `%||%` <- function (a, b) { |
|  | if (is.null(a)) b else a |
|  | } |
|  |  |
|  | `%||%` <- function(a, b) if (!is.null(a)) a else b |
|  |  |
|  | `%fin%` <- function(a, tbl) fmatch(a, tbl, 0L, NULL) > 0L |
|  |  |
|  | `%||%` <- function(x, y){ |
|  | if (is.null(x)) y else x |
|  | } |
|  |  |
|  | `%||%` <- function (lhs, rhs) { |
|  | lres <- withVisible(eval(lhs, envir = parent.frame())) |
|  | if (is.null(lres$value)) { |
|  | eval(rhs, envir = parent.frame()) |
|  | } else { |
|  | if (lres$visible) { |
|  | lres$value |
|  | } else { |
|  | invisible(lres$value) |
|  | } |
|  | } |
|  | } |
|  |  |
|  | `%&&%` <- function(lhs, rhs) { |
|  | lres <- withVisible(eval(lhs, envir = parent.frame())) |
|  | if (!is.null(lres$value)) { |
|  | eval(rhs, envir = parent.frame()) |
|  | } else { |
|  | if (lres$visible) { |
|  | lres$value |
|  | } else { |
|  | invisible(lres$value) |
|  | } |
|  | } |
|  | } |
|  |  |
|  | `%+%` <- function(x, y) { |
|  | stopifnot(is.character(x), is.character(y)) |
|  | paste0(x, y) |
|  | } |
|  |  |
|  | `%inr%` <- function(x,range) |
|  | { |
|  | if (!is.numeric(range) || length(range) != 2) |
|  | { |
|  | stop("Range must be a vector of 2 numeric values") |
|  | } |
|  | if (!is.numeric(x)) |
|  | { |
|  | stop("x must be numeric") |
|  | } |
|  | else |
|  | { |
|  | if (diff(range) < 0) |
|  | { |
|  | stop('Range must be increasing') |
|  | } |
|  |  |
|  | x >= range[1] & x <= range[2] |
|  | } |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%diag\*%` <- function(d, X) d \* X |
|  |  |
|  | `%\*diag%` <- function(X, d) t(t(X) \* d) |
|  |  |
|  | `%||%` <- function(a,b) if(is.null(a)) b else a |
|  |  |
|  | `%+%` <- function(a, b) paste0(a, b) |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x) || length(x) <= 0) { |
|  | y |
|  | } else { |
|  | x |
|  | } |
|  | } |
|  |  |
|  | `%==%` <- function(x, y) { |
|  | identical(x, y) |
|  | } |
|  |  |
|  | `%!=%` <- function(x, y) { |
|  | !identical(x, y) |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%contains%` <- function(x,y)contains(y,x) |
|  |  |
|  | `%||%` <- function(x, y) if (is.null(x)) y else x |
|  |  |
|  | `%din%` <- function(x, y) { |
|  | by <- intersect(names(x), names(y)) |
|  | nx <- nrow(x <- as.data.frame(x)) |
|  | ny <- nrow(y <- as.data.frame(y)) |
|  | bx <- x[,by,drop=FALSE] |
|  | by <- y[,by,drop=FALSE] |
|  | names(bx) = names(by) <- paste("V", seq\_len(ncol(bx)), sep="") |
|  | bz <- do.call(paste, c(rbind(bx, by), sep="\r")) |
|  | bx <- bz[seq\_len(nx)] |
|  | by <- bz[nx + seq\_len(ny)] |
|  | comm <- match(bx, by, 0) |
|  | x[comm > 0,] |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) y else x |
|  | } |
|  |  |
|  | `%nin%` <- function(x, y) { |
|  | !(x %in% y) |
|  | } |
|  |  |
|  | `%+%` <- function(lhs, rhs) { |
|  | check\_string(lhs) |
|  | check\_string(rhs) |
|  | paste0(lhs, rhs) |
|  | } |
|  |  |
|  | `%||%` <- function(lhs, rhs) { |
|  | if (!is.null(lhs)) { lhs } else { rhs } |
|  | } |
|  |  |
|  | `%s%` <- function(lhs, rhs) { |
|  | assert\_that(is.string(lhs)) |
|  | list(lhs) %>% |
|  | c(as.list(rhs)) %>% |
|  | do.call(what = sprintf) |
|  | } |
|  |  |
|  | `%+%` <- function(lhs, rhs) { |
|  | paste0(lhs, rhs) |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) { |
|  | y |
|  | } else { |
|  | x |
|  | } |
|  | } |
|  |  |
|  | `%@%` <- function(x, name) attr(x, name, exact = TRUE) |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) { |
|  | y |
|  | } else { |
|  | x |
|  | } |
|  | } |
|  |  |
|  | `%notin%` <- function(needle, haystack) { |
|  | ! (needle %in% haystack) |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%||%` <- function(a, b) if (is.null(a)) b else a |
|  |  |
|  | `%||%` <- function (a, b) if (!is.null(a)) a else b |
|  |  |
|  | `%:::%` <- function (p, f) get(f, envir = asNamespace(p)) |
|  |  |
|  | `%::%` <- function (p, f) get(f, envir = asNamespace(p)) |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) { |
|  | y |
|  | } else { |
|  | x |
|  | } |
|  | } |
|  |  |
|  | `%==%` <- function(x, y) { |
|  | identical(x, y) |
|  | } |
|  |  |
|  | `%||%` <- function(a,b) if(is.null(a)) b else a |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%:::%` <- function(p, f) { |
|  | get(f, envir = asNamespace(p)) |
|  | } |
|  |  |
|  | `%||%` <- function(a, b) { |
|  | if (is.null(a)) b else a |
|  | } |
|  |  |
|  | `%++%` <- function(l, r) { append(l, r) } |
|  |  |
|  | `%||%` <- function(a,b) if(is.null(a)) b else a |
|  |  |
|  | `%+%` <- function(a,b) paste(a, b, sep = '') |
|  |  |
|  | `%+|%` <- function(a,b) paste(a, b, sep = '|') |
|  |  |
|  | `%+&%` <- function(a,b) paste(a, b, sep = '&') |
|  |  |
|  | `%||%` <- function(l, r) { |
|  | if (is.null(l)) r else l |
|  | } |
|  |  |
|  | `%notin%` <- function(a, b) !(a %in% b) |
|  |  |
|  | `%AND%` <- function(x, y) { |
|  | if (!is.null(x) && !is.na(x)) |
|  | if (!is.null(y) && !is.na(y)) |
|  | return(y) |
|  | return(NULL) |
|  | } |
|  |  |
|  | `%AND%` <- function (x, y) |
|  | { |
|  | if (!is.null(x) && !is.na(x)) |
|  | if (!is.null(y) && !is.na(y)) |
|  | return(y) |
|  | return(NULL) |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (!is.null(x)) x else y |
|  | } |
|  |  |
|  | `%OR%` <- function(x, y) { |
|  | if (is.null(x) || isTRUE(is.na(x))) |
|  | y |
|  | else |
|  | x |
|  | } |
|  |  |
|  | `%AND%` <- function(x, y) { |
|  | if (!is.null(x) && !is.na(x)) |
|  | if (!is.null(y) && !is.na(y)) |
|  | return(y) |
|  | return(NULL) |
|  | } |
|  |  |
|  | `%.%` <- function(x, y) { |
|  | paste(x, y, sep='') |
|  | } |
|  |  |
|  | `%OR%` <- function(x, y) { |
|  | if (is.null(x) || isTRUE(is.na(x))) |
|  | y |
|  | else |
|  | x |
|  | } |
|  |  |
|  | `%AND%` <- function(x, y) { |
|  | if (!is.null(x) && !is.na(x)) |
|  | if (!is.null(y) && !is.na(y)) |
|  | return(y) |
|  | return(NULL) |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%||%` <- function(a, b) { |
|  | if (is.null(a)) b else a |
|  | } |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x)) y else x |
|  | } |
|  |  |
|  | `%identical%` <- function(x, y) identical(x, y) |
|  |  |
|  | `%assert\_class%` <- function(dat, class){ |
|  | assert\_class(dat = dat, class = class) |
|  | } |
|  |  |
|  | `%||%` <- function(a, b) if (is.null(a)) b else a |
|  |  |
|  | `%||%` <- function(x, y) { |
|  | if (is.null(x) || length(x) == 0L) y else x |
|  | } |
|  |  |
|  | `%+%` <- function(chr1, chr2){ |
|  | paste0(chr1, chr2) |
|  | } |
|  |  |
|  | `%||%` <- function(l, r) if (is.null(l)) r else l |
|  |  |
|  | `%+%` <- function(l, r) { |
|  | assert\_string(l) |
|  | assert\_string(r) |
|  | paste0(l, r) |
|  | } |
|  |  |
|  | `%||%` <- function(a, b) if (is.null(a)) b else a |
|  |  |

[**try-this-at-home.R**](https://gist.github.com/hrbrmstr/58b827dea95a7adf0f0e751b44b75b0c#file-try-this-at-home-r)

|  |  |
| --- | --- |
|  | #' --- |
|  | #' title: "" |
|  | #' author: "" |
|  | #' date: "" |
|  | #' output: |
|  | #' html\_document: |
|  | #' keep\_md: true |
|  | #' theme: simplex |
|  | #' highlight: monochrome |
|  | #' --- |
|  | #+ init, include=FALSE |
|  | knitr::opts\_chunk$set(message = FALSE, warning = FALSE, dev="png", |
|  | fig.retina = 2, fig.width = 10, fig.height = 6) |
|  |  |
|  | #+ libs |
|  | library(stringi) |
|  | library(tidyverse) |
|  |  |
|  | #+ data, cache=TRUE |
|  | options(width=120) |
|  | utils <- read\_rds(url("https://rud.is/dl/utility-belt.rds")) |
|  |  |
|  |  |
|  | #+ filter |
|  | utils |
|  |  |
|  | filter(utils, stri\_detect\_fixed(file\_src, "`%")) %>% # only find sources with infix definitions |
|  | pull(file\_src) %>% |
|  | paste0(collapse="\n\n") %>% |
|  | parse(text = ., keep.source=TRUE) -> infix\_src |
|  |  |
|  | str(infix\_src, 1) |
|  |  |
|  | infix\_parsed <- tbl\_df(getParseData(infix\_src)) |
|  |  |
|  | infix\_parsed |
|  |  |
|  | # pattern for function definition |
|  | pat <- c("SYMBOL", "expr", "LEFT\_ASSIGN", "expr", "FUNCTION") |
|  |  |
|  | # find all of ^^ sequences (there's a good twitter discussion abt a month ago on this if you can find it) |
|  | idx <- which(infix\_parsed$token == pat[1]) # find location of match of start of seq |
|  |  |
|  | # look for the rest of the sequences starting at each idx position |
|  | map\_lgl(idx, ~{ |
|  | all(infix\_parsed$token[.x:(.x+(length(pat)-1))] == pat) |
|  | }) -> found |
|  |  |
|  | f\_defs <- idx[found] # starting indices of all the places where functions are defined |
|  |  |
|  | # filter ^^ to only find infix ones |
|  | infix\_defs <- f\_defs[stri\_detect\_regex(infix\_parsed$text[f\_defs], "^`\\%")] |
|  |  |
|  | # there aren't too many, but remember we're just searching `util` functions |
|  | length(infix\_defs) |
|  |  |
|  | # nuke a file and fill it with the function definition |
|  | cat("", sep="", file="infix\_functions.R") |
|  | walk2( |
|  | getParseText(infix\_parsed, infix\_parsed$id[infix\_defs]), # extract the infix name |
|  | getParseText(infix\_parsed, infix\_parsed$id[infix\_defs + 3]), # extract the function definition body |
|  | ~{ |
|  | cat(.x, " <- ", .y, "\n\n", sep="", file="infix\_functions.R", append=TRUE) |
|  | } |
|  | ) |

Here's an overview of what you can expect to find:

# A tibble: 39 x 2

name n

1 `%||%` 47

2 `%+%` 7

3 `%AND%` 4

4 `%notin%` 4

5 `%:::%` 3

6 `%==%` 3

7 `%!=%` 2

8 `%\*diag%` 2

9 `%diag\*%` 2

10 `%nin%` 2

11 `%OR%` 2

12 `%::%` 1

13 `%??%` 1

14 `%.%` 1

15 `%@%` 1

16 `%&&%` 1

17 `%&%` 1

18 `%+&%` 1

19 `%++%` 1

20 `%+|%` 1

21 `%<<%` 1

22 `%>>%` 1

23 `%~~%` 1

24 `%assert\_class%` 1

25 `%contains%` 1

26 `%din%` 1

27 `%fin%` 1

28 `%identical%` 1

29 `%In%` 1

30 `%inr%` 1

31 `%M%` 1

32 `%notchin%` 1

33 `%or%` 1

34 `%p%` 1

35 `%pin%` 1

36 `%R%` 1

37 `%s%` 1

38 `%sub\_in%` 1

39 `%sub\_nin%` 1