**Do more with dates and times in R with lubridate 1.3.0**

*note: This vignette is an updated version of the blog post first published at*[*r-statistics*](http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/)

Lubridate is an R package that makes it easier to work with dates and times. Below is a concise tour of some of the things lubridate can do for you.

Lubridate was created by Garrett Grolemund and Hadley Wickham.

**Parsing dates and times**

Getting R to agree that your data contains the dates and times you think it does can be tricky. Lubridate simplifies that. Identify the order in which the year, month, and day appears in your dates. Now arrange “y”, “m”, and “d” in the same order. This is the name of the function in lubridate that will parse your dates. For example,

library(lubridate)

ymd("20110604")

## [1] "2011-06-04 UTC"

mdy("06-04-2011")

## [1] "2011-06-04 UTC"

dmy("04/06/2011")

## [1] "2011-06-04 UTC"

Lubridate's parse functions handle a wide variety of formats and separators, which simplifies the parsing process.

If your date includes time information, add h, m, and/or s to the name of the function. ymd\_hms is probably the most common date time format. To read the dates in with a certain time zone, supply the official name of that time zone in the tz argument.

arrive <- ymd\_hms("2011-06-04 12:00:00", tz = "Pacific/Auckland")

arrive

## [1] "2011-06-04 12:00:00 NZST"

leave <- ymd\_hms("2011-08-10 14:00:00", tz = "Pacific/Auckland")

leave

## [1] "2011-08-10 14:00:00 NZST"

**Setting and Extracting information**

Extract information from date times with the functions second, minute, hour, day, wday, yday, week, month, year, and tz. You can also use each of these to set (i.e, change) the given information. Notice that this will alter the date time. wday and month have an optional label argument, which replaces their numeric output with the name of the weekday or month.

second(arrive)

## [1] 0

second(arrive) <- 25

arrive

## [1] "2011-06-04 12:00:25 NZST"

second(arrive) <- 0

wday(arrive)

## [1] 7

wday(arrive, label = TRUE)

## [1] Sat

## Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

**Time Zones**

There are two very useful things to do with dates and time zones. First, display the same moment in a different time zone. Second, create a new moment by combining an existing clock time with a new time zone. These are accomplished by with\_tz andforce\_tz.

For example, a while ago I was in Auckland, New Zealand. I arranged to meet the co-author of lubridate, Hadley, over skype at 9:00 in the morning Auckland time. What time was that for Hadley who was back in Houston, TX?

meeting <- ymd\_hms("2011-07-01 09:00:00", tz = "Pacific/Auckland")

with\_tz(meeting, "America/Chicago")

## [1] "2011-06-30 16:00:00 CDT"

So the meetings occurred at 4:00 Hadley's time (and the day before no less). Of course, this was the same actual moment of time as 9:00 in New Zealand. It just appears to be a different day due to the curvature of the Earth.

What if Hadley made a mistake and signed on at 9:00 his time? What time would it then be my time?

mistake <- force\_tz(meeting, "America/Chicago")

with\_tz(mistake, "Pacific/Auckland")

## [1] "2011-07-02 02:00:00 NZST"

His call would arrive at 2:00 am my time! Luckily he never did that.

**Time Intervals**

You can save an interval of time as an Interval class object with lubridate. This is quite useful! For example, my stay in Auckland lasted from June 4, 2011 to August 10, 2011 (which we've already saved as arrive and leave). We can create this interval in one of two ways:

auckland <- interval(arrive, leave)

auckland

## [1] 2011-06-04 12:00:00 NZST--2011-08-10 14:00:00 NZST

auckland <- arrive %--% leave

auckland

## [1] 2011-06-04 12:00:00 NZST--2011-08-10 14:00:00 NZST

My mentor at the University of Auckland, Chris, traveled to various conferences that year including the Joint Statistical Meetings (JSM). This took him out of the country from July 20 until the end of August.

jsm <- interval(ymd(20110720, tz = "Pacific/Auckland"), ymd(20110831, tz = "Pacific/Auckland"))

jsm

## [1] 2011-07-20 NZST--2011-08-31 NZST

Will my visit overlap with and his travels? Yes.

“{r] int\_overlaps(jsm, auckland)

Then I better make hay while the sun shines! For what part of my visit will Chris be there?

```r

setdiff(auckland, jsm)

## [1] 2011-06-04 12:00:00 NZST--2011-07-20 NZST

Other functions that work with intervals include int\_start, int\_end, int\_flip, int\_shift, int\_aligns, union, intersect, setdiff, and %within%.

**Arithmetic with date times**

Intervals are specific time spans (because they are tied to specific dates), but lubridate also supplies two general time span classes: Durations and Periods. Helper functions for creating periods are named after the units of time (plural). Helper functions for creating durations follow the same format but begin with a "d” (for duration) or, if you prefer, and “e” (for exact).

minutes(2) ## period

## [1] "2M 0S"

dminutes(2) ## duration

## [1] "120s (~2 minutes)"

Why two classes? Because the timeline is not as reliable as the number line. The Duration class will always supply mathematically precise results. A duration year will always equal 365 days. Periods, on the other hand, fluctuate the same way the timeline does to give intuitive results. This makes them useful for modeling clock times. For example, durations will be honest in the face of a leap year, but periods may return what you want:

leap\_year(2011) ## regular year

## [1] FALSE

ymd(20110101) + dyears(1)

## [1] "2012-01-01 UTC"

ymd(20110101) + years(1)

## [1] "2012-01-01 UTC"

leap\_year(2012) ## leap year

## [1] TRUE

ymd(20120101) + dyears(1)

## [1] "2012-12-31 UTC"

ymd(20120101) + years(1)

## [1] "2013-01-01 UTC"

You can use periods and durations to do basic arithmetic with date times. For example, if I wanted to set up a reoccuring weekly skype meeting with Hadley, it would occur on:

meetings <- meeting + weeks(0:5)

Hadley travelled to conferences at the same time as Chris. Which of these meetings would be affected? The last two.

meetings %within% jsm

## [1] FALSE FALSE FALSE TRUE TRUE TRUE

How long was my stay in Auckland?

auckland/edays(1)

## [1] 67.08

auckland/edays(2)

## [1] 33.54

auckland/eminutes(1)

## [1] 96600

And so on. Alternatively, we can do modulo and integer division. Sometimes this is more sensible than division - it is not obvious how to express a remainder as a fraction of a month because the length of a month constantly changes.

auckland%/%months(1)

## [1] 2

auckland%%months(1)

## [1] 2011-08-04 12:00:00 NZST--2011-08-10 14:00:00 NZST

Modulo with an timespan returns the remainder as a new (smaller) interval. You can turn this or any interval into a generalized time span with as.period.

as.period(auckland%%months(1))

## [1] "6d 2H 0M 0S"

as.period(auckland)

## [1] "2m 6d 2H 0M 0S"

**If anyone drove a time machine, they would crash**

The length of months and years change so often that doing arithmetic with them can be unintuitive. Consider a simple operation, January 31st + one month. Should the answer be

1. February 31st (which doesn't exist)
2. March 4th (31 days after January 31), or
3. February 28th (assuming its not a leap year)

A basic property of arithmetic is that a + b - b = a. Only solution 1 obeys this property, but it is an invalid date. I've tried to make lubridate as consistent as possible by invoking the following rule *if adding or subtracting a month or a year creates an invalid date, lubridate will return an NA*. This is new with version 1.3.0, so if you're an old hand with lubridate be sure to remember this!

If you thought solution 2 or 3 was more useful, no problem. You can still get those results with clever arithmetic, or by using the special %m+% and %m-% operators. %m+% and %m-% automatically roll dates back to the last day of the month, should that be necessary.

jan31 <- ymd("2013-01-31")

jan31 + months(0:11)

## [1] "2013-01-31 UTC" NA "2013-03-31 UTC" NA

## [5] "2013-05-31 UTC" NA "2013-07-31 UTC" "2013-08-31 UTC"

## [9] NA "2013-10-31 UTC" NA "2013-12-31 UTC"

floor\_date(jan31, "month") + months(0:11) + days(31)

## [1] "2013-02-01 UTC" "2013-03-04 UTC" "2013-04-01 UTC" "2013-05-02 UTC"

## [5] "2013-06-01 UTC" "2013-07-02 UTC" "2013-08-01 UTC" "2013-09-01 UTC"

## [9] "2013-10-02 UTC" "2013-11-01 UTC" "2013-12-02 UTC" "2014-01-01 UTC"

jan31 %m+% months(0:11)

## [1] "2013-01-31 UTC" "2013-02-28 UTC" "2013-03-31 UTC" "2013-04-30 UTC"

## [5] "2013-05-31 UTC" "2013-06-30 UTC" "2013-07-31 UTC" "2013-08-31 UTC"

## [9] "2013-09-30 UTC" "2013-10-31 UTC" "2013-11-30 UTC" "2013-12-31 UTC"

Notice that this will only affect arithmetic with months (and arithmetic with years if your start date it Feb 29).

**Vectorization**

The code in lubridate is vectorized and ready to be used in both interactive settings and within functions. As an example, I offer a function for advancing a date to the last day of the month

last\_day <- function(date) {

ceiling\_date(date, "month") - days(1)

}

**Further Resources**

To learn more about lubridate, including the specifics of periods and durations, please read the [original lubridate paper](http://www.jstatsoft.org/v40/i03/). Questions about lubridate can be addressed to the lubridate google group. Bugs and feature requests should be submitted to the [lubridate development page](http://github.com/hadley/lubridate) on github.