Dates and times with lubridate :: CHEAT SHEET

### Dates and Times

**Date-times**

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2017-11-28&quot;</td>
<td>&quot;12:00:00&quot;</td>
<td>A date-time is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC.</td>
</tr>
</tbody>
</table>

**Parse Date-times** (Convert strings or numbers to date-times)

1. Identify the order of the year (y), month (m), day (d), hour (h), minute (m) and second (s) elements in your data.
2. Use the function whose name replicates the order. Each accepts a tz argument to set the time zone, e.g. ymd(x, tz = "UTC").

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymd_hms()</td>
<td>YMD-HMS: All of the above</td>
</tr>
<tr>
<td>ymd_h()</td>
<td>YMD: YMD, H: hours, M: minutes, S: seconds</td>
</tr>
<tr>
<td>mdy_hms()</td>
<td>MDY-HMS: M: month (1-12), D: day, S: seconds</td>
</tr>
<tr>
<td>mdy_h()</td>
<td>MDY: M: month (1-12), D: day, H: hours, M: minutes</td>
</tr>
<tr>
<td>dmy_hms()</td>
<td>DMY-HMS: D: day, M: month (1-12), S: seconds</td>
</tr>
<tr>
<td>dmy_h()</td>
<td>DMY: D: day, M: month (1-12), H: hours, M: minutes</td>
</tr>
<tr>
<td>ymd()</td>
<td>YMD: YMD, H: hours, M: minutes</td>
</tr>
<tr>
<td>ymdl()</td>
<td>YMDL: YMD, L: leap year</td>
</tr>
<tr>
<td>ymdhms()</td>
<td>YMD-HMS: All of the above</td>
</tr>
<tr>
<td>ymdh()</td>
<td>YMD: YMD, H: hours, M: minutes</td>
</tr>
<tr>
<td>mdyhs()</td>
<td>MDY-HS: M: month (1-12), D: day, S: seconds</td>
</tr>
<tr>
<td>mdyh()</td>
<td>MDY: M: month (1-12), D: day, H: hours, M: minutes</td>
</tr>
<tr>
<td>dmyhs()</td>
<td>DMY-HS: D: day, M: month (1-12), S: seconds</td>
</tr>
<tr>
<td>dmyh()</td>
<td>DMY: D: day, M: month (1-12), H: hours, M: minutes</td>
</tr>
</tbody>
</table>

**Get and Set Components**

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date()</td>
<td>Date component. date(dt)</td>
</tr>
<tr>
<td>year()</td>
<td>Year of date. year(dt)</td>
</tr>
<tr>
<td>isoyear()</td>
<td>The ISO 8601 year. isoyear(dt)</td>
</tr>
<tr>
<td>epiyear()</td>
<td>Epidemiological year. epiyear(dt)</td>
</tr>
<tr>
<td>month()</td>
<td>Month of date. month(dt)</td>
</tr>
<tr>
<td>wday()</td>
<td>Day of the week. wday(dt)</td>
</tr>
<tr>
<td>qday()</td>
<td>Day of the quarter. qday(dt)</td>
</tr>
<tr>
<td>hour()</td>
<td>Hour of date. hour(dt)</td>
</tr>
<tr>
<td>minute()</td>
<td>Minutes of date. minute(dt)</td>
</tr>
<tr>
<td>second()</td>
<td>Seconds of date. second(dt)</td>
</tr>
<tr>
<td>tz()</td>
<td>Time zone. tz(dt)</td>
</tr>
<tr>
<td>week()</td>
<td>Week of the year. week(dt)</td>
</tr>
<tr>
<td>isoweek()</td>
<td>ISO 8601 week. isoweek(dt)</td>
</tr>
<tr>
<td>epiweek()</td>
<td>Epidemiological week. epiweek(dt)</td>
</tr>
<tr>
<td>quarter()</td>
<td>Quarter of the year. quarter(dt)</td>
</tr>
<tr>
<td>semester()</td>
<td>Semester of the year. semester(dt)</td>
</tr>
<tr>
<td>leap_year()</td>
<td>Is it a leap year? leap_year(dt)</td>
</tr>
<tr>
<td>update()</td>
<td>(object, ..., simple = FALSE)</td>
</tr>
</tbody>
</table>

**Examples**

- 2017-11-28 12:00:00
- 2017-11-28

**Round Date-times**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>round_date()</td>
<td>Round to nearest unit. round_date(dt, unit = &quot;month&quot;)</td>
</tr>
<tr>
<td>ceiling_date()</td>
<td>Round up to nearest unit. ceiling_date(dt, unit = &quot;month&quot;)</td>
</tr>
</tbody>
</table>

**Stamp Date-times**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp()</td>
<td>Derive a template from an example string and return a new function that will apply the template to date-times. Also stamp_date() and stamp_time().</td>
</tr>
</tbody>
</table>

**Examples**

1. Derive a template, create a function
   `sf = stamp("Created Sunday, Jan 17, 1999 3:34")`  
   `## [1] "Created Monday, Apr 05, 2010 00:00:00"`

2. Apply the template to dates
   `stf(ymd("2010-04-05"))`  
   `## [1] "Created Monday, Apr 05, 2010 00:00:00"`

**Time Zones**

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns one time zone per vector.

Use the UTC time zone to avoid Daylight Savings.

**Olson Names**

Returns a list of valid time zone names.

**Sys.timezone**

Gets current time zone.

**Examples**

- 2017-11-28 12:00:00
- 2017-11-28

---

**RStudio**

RStudio® is a trademark of RStudio, PBC.  •  CC BY-SA  RStudio  •  info@rstudio.com  •  844-448-1212  •  rstudio.com  •  Learn more at lubridate.tidyverse.org  •  lubridate: 1.7.10  •  Updated: 2021-07
Math with Date-times | Lubridate provides three classes of timespans to facilitate math with dates and date-times.

## Periods

Track changes in clock times, which ignore time line irregularities.

- **A normal day**
  - `nor <- ymd_hms("2018-01-01 01:30:00","z="US/Eastern")`

- **The start of daylight savings (spring forward)**
  - `gap <- ymd_hms("2018-03-11 01:30:00","z="US/Eastern")`

- **The end of daylight savings (fall back)**
  - `lap <- ymd_hms("2018-11-04 01:30:00","z="US/Eastern")`

- **Leap years and leap seconds**
  - `leap <- ymd("2019-09-01")`

**Examples**

- `nor + minutes(90)`
- `gap + minutes(90)`
- `lap + minutes(90)`
- `leap + years(1)`
- `leap + dyears(1)`

## Durations

Track the passage of physical time, which deviates from clock time when irregularities occur.

- **Example**
  - `i <- interval(ymd("2017-01-01"), d)`
  - `int_length(i)`
  - `int_aligns(i, j)`
  - `int_shiftdays(i, j)`
  - `as.dtime(i)`
  - `make_dtime(x)`

## Intervals

Represent specific intervals of the timeline, bounded by start and end date-times.

- **Example**
  - `i <- interval(ymd("2017-01-01"), d)`
  - `int_length(i)`
  - `int_aligns(i, j)`
  - `int_shiftdays(i, j)`
  - `as.dtime(i)`

**Examples**

- `i <- interval(ymd("2017-01-01"), d)`
  - `int_length(i)`
  - `int_aligns(i, j)`
  - `int_shiftdays(i, j)`
  - `as.dtime(i)`

- `gap + minutes(90)`
- `nor + minutes(90)`
- `lap + minutes(90)`
- `leap + years(1)`

## Building Timespans

Lubridate provides three classes of timespans to facilitate math with dates and date-times.

- `period()`
  - `period(x, unit = "second")`
  - `period(x, unit = "second")`

- `duration()`
  - `duration(num = NULL, units = "second", ...)`

- `int_length(i)`

- `as.period(x, unit = "second")` | `as.period(i)`

- `period_to_seconds(x)` | `period_to_seconds(i)`

**Examples**

- `period(5, unit = "years")`
- `period(99999)`

**Mathematical Functions**

- `period_to_seconds(p)`
  - `by the period. Also` | `period, optionally in the specified units.`

- `period(x, unit = "second")`
  - `x = 1`

- `number(x)`
  - `x = 1`

- `num = NULL, units = "second" , ...`

- `x, num = NULL, units = "second" , ...`

- `x, num = NULL, units = "second" , ...`

- `as.duration(i)`

**Note**

- Not all years are 365 days due to leap days.
- Not all minutes are 60 seconds due to leap seconds.

- It is possible to create an imaginary date by adding months, e.g. February 31st
  - `jan31 <- ymd(20180131)`
  - `jan31 + months(1)`
  - `# NA`
  - `%m%-%m%` and `%m%-%m%` will roll imaginary dates to the last day of the previous month.
  - `jan31+%m%+months(1)`
  - `# 2018-02-28`

- `add_with_rollback(e1, e2, roll_to_first = TRUE)` will roll imaginary dates to the first day of the new month.

- `add_with_rollback(jan31, months(1), roll_to_first = TRUE)`
  - `# 2018-03-01`

---

**RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at lubridate.tidyverse.org • lubridate 1.7.10 • Updated: 2021-07**