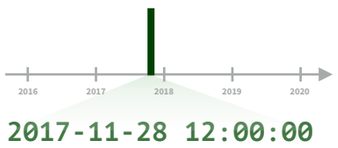
Fechas and tiempos con lubridate : : **GUÍA RÁPIDA**

date-times



2017-11-28 12:00:00

2017-11-28 12:00:00
Un **date-time** es un punto en el tiempo, almacenado como el número de segundos desde 1970-01-01 00:00:00 UTC

dt <- ***as_datetime****(1511870400)*
"2017-11-28 12:00:00 UTC"

2017-11-28
Un **date** es un día almacenado cómo el número de días desde 1970-01-01

d <- ***as_date****(17498)*
"2017-11-28"

12:00:00
Un hms es un **time** almacenado como el número de segundos desde 00:00:00

*t <- hms::****as.hms****(85)*
00:01:25

FILTRA DATE-TIMES (Convierte cadenas o números a date-times)

1. Identifica el orden del año (**y**), mes (**m**), día (**d**), hora (**h**), minuto (**m**) y segundo (**s**) en tus datos
2. Usa la función inferior cuyo nombre reproduce el orden de tus datos.Cada una acepta una amplia variedad de formatos de entrada.

2017-11-28T14:02:00	**ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*
2017-22-12 10:00:00	**ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*
11/28/2017 1:02:03	**mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*
1 Jan 2017 23:59:59	**dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*
20170131	**ymd**(), **ydm**(). *ymd(20170131)*
July 4th, 2000	**mdy**(), **myd**(). *mdy("July 4th, 2000")*
4th of July '99	**dmy**(), **dym**(). *dmy("4th of July '99")*
2001: Q3	**yq**() Q para el trimestre. *yq("2001: Q3")*
2:01	hms::**hms**() También lubridate::**hms**(), **hm**() and **ms**(), que devuelve periodos.* *hms::hms(sec = 0, min= 1, hours = 2)*
2017.5	**date_decimal**(decimal, tz = "UTC") Q para trimestre. *date_decimal(2017.5)*
	now(tzone = "") Hora actual en tz (por defecto al valor del sistema tz). *now()*
	today(tzone = "") Fecha actual en tz (por defecto al valor del sistema tz). *today()*
	fast_strptime() strptime rápido. *fast_strptime('9/1/01', '%y/%m/%d')*
	parse_date_time() strptime fácil. *parse_date_time("9/1/01", "ymd")*

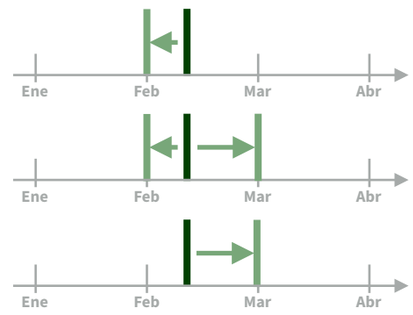
CONSIGUE Y DEFINE COMPONENTES

Usa una función para conseguir un componente. *d ## "2017-11-28"*
day(d) ## 28

Asigna a una función para cambiar un componente. *day(d) <- 1*
d ## "2017-11-01"

2018-01-31 11:59:59	**date**(x) Componente fecha. *date(dt)*
2018-01-31 11:59:59	**year**(x) Year. *year(dt)* **isoyear**(x) El año ISO 8601. **epiyear**(x) Año epidemiológico.
2018-01-31 11:59:59	**month**(x, label, abbr) Mes. *month(dt)*
2018-01-31 11:59:59	**day**(x) Día del mes. *day(dt)* **wday**(x,label,abbr) Día de la semana. **qday**(x) Día del trimestre.
2018-01-31 11:59:59	**hour**(x) Hora. *hour(dt)*
2018-01-31 11:59:59	**minute**(x) Minutos. *minute(dt)*
2018-01-31 11:59:59	**second**(x) Segundos. *second(dt)*
	week(x) Semana del año. *week(dt)* **isoweek**() SemanaISO 8601. **epiweek**() Semana epidemiológica.
	quarter(x, with_year = FALSE) Trimestre. *quarter(dt)*
	semester(x, with_year = FALSE) Semestre. *semester(dt)*
	am(x) ¿Es am? *am(dt)* **pm**(x) ¿Es pm? *pm(dt)*
	dst(x) ¿Es horario de verano? *dst(d)*
	leap_year(x) ¿Es año bisiesto? *leap_year(d)*
	update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Redondear Date-times



floor_date(x, unit = "second") Redondear hacia la unidad más cercana inferior. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Redondear a la unidad más cercana. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Redondear hacia la unidad más cercana superior. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Devuelve el ultimo día del mes previo. *rollback(dt)*

Stamp Date-times

stamp() Genera una plantilla de una cadena de ejemplo y devolve una nueva función que aplicará la plantilla a date-times. Además **stamp_date**() y **stamp_time**().

1. Genera una plantilla, crea una función
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Aplica la plantilla a fechas
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

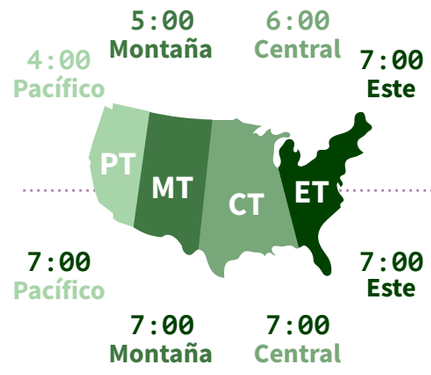
Tip: use a date with day > 12

Zonas horarias

R reconoce ~600 zonas horarias. Cada una incluye la zona horaria, Horario de verano, variaciones históricas del calendario para un área. R asigna *una zona horaria* por vector.

Usa la zona horaria **UTC** para evitar el Horario de Verano.

OlsonNames() Devuelve una lista de nombres válidos de zonas horarias. *OlsonNames()*



with_tz(time, tzone = "") Consigue la misma **misma date-time** en una nueva zona horaria (un nuevo huso horario). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Consigue el mismo et the **mismo huso horario** en una zona horaria (una nueva zona horaria). *force_tz(dt, "US/Pacific")*

Matemáticas con Date-times — Lubridate proporciona tres clases de duraciones para facilitar las operaciones con fechas y date-times

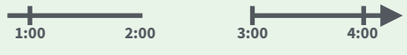
Las operaciones con date-times se basa en un **intervalo temporal**, que se comporta inconsistentemente. Considera como un intervalo temporal se comporta durante:

Un día normal

nor <- ymd_hms("2018-01-01 01:30:00",tz="US/Eastern")

El comienzo del horario de verano (primavera en adelante)

gap <- ymd_hms("2018-03-11 01:30:00",tz="US/Eastern")

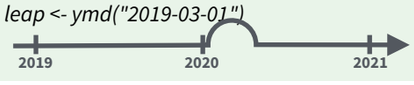


El final del horario de verano (vuelta atrás)

lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")

Años bisiestos y segundos bisiestos

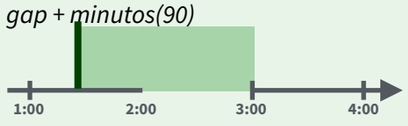
leap <- ymd("2019-03-01")



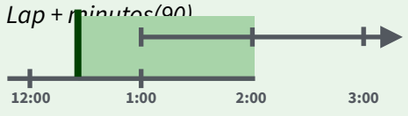
Periodos sigue los cambios en las horas, ignorando irregularidades en el intervalo temporal.

normal + minutos(90)

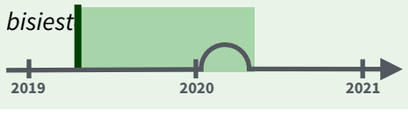
gap + minutos(90)



Lap + minutos(90)

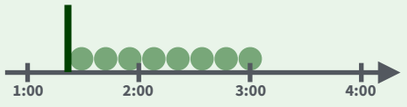


bisiesto + dyears(1)

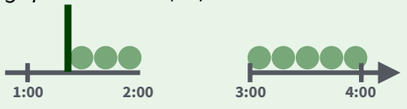


Duraciones sigue el paso del tiempo físico, que se desvía del tiempo de reloj candy aparecen irregularidades.

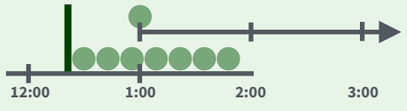
normal + dminutos(90)



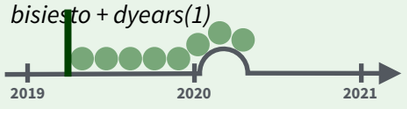
gap + dminutos(90)



lap+ dminutes(90)

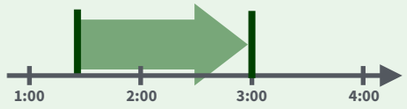


bisiesto + dyears(1)

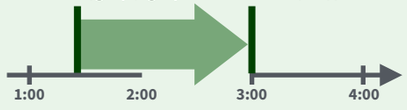


Intervalos representan intervalos específicos de un intervalo temporal, delimitado por un comienzo y un final date-times.

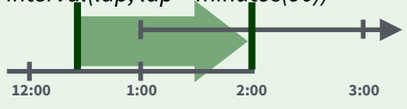
interval(normal, normal + minutos(90))



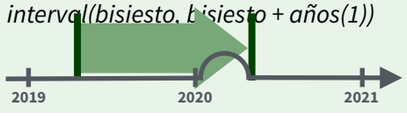
interval(gap, gap + minutos(90))



interval(lap, lap + minutos(90))



interval(bisiesto, bisiesto + años(1))



No todos los años tienen 365 días por los **días bisiestos.**

No todos los minutos Tienen 60 segundos debido a los **minutos bisiestos**.

Es posible crear fechas imaginarias añadiendo **meses**, e.g. Febrero 31

jan31 <- ymd(20180131)
jan31 + months(1)
NA

%m+% y **%m-%** añadirán fechas imaginarias al último día del mes previo.

jan31 %m+% months(1)
"2018-02-28"

add_with_rollback(e1, e2, roll_to_first = TRUE) añadirán fechas al primer día del nuevo mes.

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

PERIODOS

Añade o resta periodos para modelizar eventos que ocurren en momentos específicos de reloj, como la apertura de la sesión del NYSE.

Crea un periodo con el nombre de la unidad de tiempo en **plural**, e.g.

p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"

Número de meses | Número de días | etc.

years(x = 1) x años.
months(x) x meses.
weeks(x = 1) x semanas.
days(x = 1) x días.
hours(x = 1) x horas.
minutes(x = 1) x minutos.
seconds(x = 1) x segundos.
milliseconds(x = 1) x millisegundos.
microseconds(x = 1) x microsegundos
nanoseconds(x = 1) x millisegundos.
picoseconds(x = 1) x picosegundos.

period(num = NULL, units = "second", ...) Un constructor automático de periodos amigable. *period(5, unit = "years")*

as.period(x, unit) Transforma un intervalo temporal a un periodo, opcionalmente en las unidades especificadas. También **is.period**(). *as.period(i)*

period_to_seconds(x) Convierte un periodo al número “estándar” de segundos del periodo. También **seconds_to_period**(). *period_to_seconds(p)*

DURACIONES

Añade o resta duraciones para modelizar procesos físicos, como la duración de la batería. Las duraciones se guardan como segundos, la única unidad temporal con una longitud consistente. **Difftimes** son una clase de duraciones disponibles en R base.

Crea una duración con el nombre del periodo con el prefijo ***d***, e.g.

dd <- ddays(14)
dd
"1209600s (~2 weeks)"

Longitud exacta en segundos | Equivalente en unidades comunes

dyears(x = 1) 31536000x segundos.
dweeks(x = 1) 604800x segundos.
ddays(x = 1) 86400x segundos.
dhours(x = 1) 3600x segundos.
dminutes(x = 1) 60x segundos.
dseconds(x = 1) x segundos.
dmilliseconds(x = 1) x $\times 10^{-3}$ segundos.
dmicroseconds(x = 1) x $\times 10^{-6}$ segundos.
dnanoseconds(x = 1) x $\times 10^{-9}$ segundos.
dpicoseconds(x = 1) x $\times 10^{-12}$ segundos.

duration(num = NULL, units = "second", ...) Un constructor automático de periodos amigable. *duration(5, unit = "years")*

as.duration(x, …) Transforma un intervalo temporal a una duración. También **is.duration**(), **is.difftime**(). *as.duration(i)*

make_difftime(x) Crea difftime con el número especificado de unidades. *make_difftime(99999)*

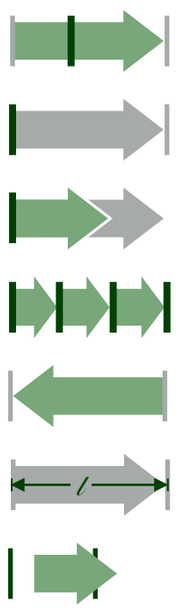
INTERVALOS

Divide un intervalo por una duración para determinar su longitud física, divide un intervalo por un periodo para determinar su longitud en unidades de reloj.

Crea un intervalo con **interval**() or **%--%**, e.g.

Inicio | Final

*i <- **interval**(ymd("2017-01-01"), d) ## 2017-01-01 UTC--2017-11-28 UTC*
*j <- d **%--%** ymd("2017-12-31") ## 2017-11-28 UTC--2017-12-31 UTC*



a **%within%** b Cae el intervalo o date-time en el intervalo *a* en el *b*? *now() %within% i*

int_start(int) Accede/define el inicio del date-time de un intervalo. Además **int_end**(). *int_start(i) <- now(); int_start(i)*

int_aligns(int1, int2) ¿Comparten límites los dos intervalos? También **int_overlaps**(). *int_aligns(i, j)*

int_diff(times) Crea los intervalos que ocurren entre date-times en un vector. *v <-c(dt, dt + 100, dt + 1000)); int_diff(v)*

int_flip(int) Cambia la dirección de un intervalo. También **int_standardize**(). *int_flip(i)*

int_length(int) Longitud en segundos. *int_length(i)*

int_shift(int, by) Mueve un intervalo adelante/ atrás por un intervalo de tiempo. *int_shift(i, days(-1))*

as.interval(x, start, …) Transforma un intervalo de tiempo a un intervalo con un inicio. También **is.interval**(). *as.interval(days(1), start = now())*