time and Unix

background

humans vs time

Our understanding of calendars and dates has changed markedly throughout recorded history.

- https://en.wikipedia.org/wiki/Calendar_era
- https://en.wikipedia.org/wiki/List_of_calendars

Being a human construct, calendars are notoriously subject to political manipulation and change. Without going into detail, making changes to calendars to fix even egregious known bugs are still momentuous disruptions for society when they occur and they leave their mark in history.

- https://en.wikipedia.org/wiki/Adoption_of_the_Gregorian_calendar
- https://en.wikipedia.org/wiki/List_of_adoption_dates_of_the_Gregorian_calendar_by_country

Managing time generally at scales smaller than a day famously didn't really need coordination until the invention of the railway. Later, the notion of Daylight Savings Time were proposed and adopted, adding to the complexity.

- https://en.wikipedia.org/wiki/Time_zone
- https://en.wikipedia.org/wiki/Daylight_saving_time
- https://en.wikipedia.org/wiki/24-hour_clock

Astronomers have had to manage different definitions for time, finding that the day as measured by the Sun would tend to process over successive days due to changing position of the Earth and Sun throughout the year.

- https://en.wikipedia.org/wiki/Sidereal_time

When atomic clocks were invented it became clear that the Earth did not rotate on a regular period and fixing the length of a day also needed adjusting every so often. So far, the Earth appears to have slowed slightly and the leap second adjustments have been positive. There are signs that we might be coming up to needing a negative leap second at some point in the next few years which will be interesting to watch as there is a push to have general timekeeping do away with the leap second altogether and stick to the TAI definition of time, based on the atomic clock.

- https://en.wikipedia.org/wiki/Leap_second

More recently, scientists have had to create calendars for use on other astronomical bodies

- https://en.wikipedia.org/wiki/Timekeeping_on_the_Moon
- https://en.wikipedia.org/wiki/Timekeeping_on_Mars

computers vs time

The relationship between computer clocks and time started small, localised and without context.

Computers have traditionally used a number of different timers for various internal functions. These manage multiple processes and subsystems in a way that their actions are coordinated, but generally without any reference to the outside world.

When systems did begin to have their own clock, they relied on being manually set at boot and worked by counting their internal timer ticks to keep pace as long as they had power. Battery-backed clocks keep reasonably accurate time but still tend to drift, forcing regular adjustment.

As with general timekeeping, this computer clock inaccuracy wasn't a problem until they are compared to others and it becomes obvious that they need to be kept in some kind of sync.

Initially, organisations like NIST and CSIRO maintained accurate time and broadcast these via radio to consumers. As computer networks became more ubiquitous, the Network Time Protocol was used to understand the relative difference between system clocks and improve accuracy further.

Later, cheap GPS receivers became commonly used to provide very good sources of time discipline for consumer devices.

computers vs humans

So we have a complex set of human-scale date- and time-setting rules to follow and computers with high-precision monotonic clocks, now it's time to combine these

Unix time

In the beginning Unix systems traditionally used a clock that recorded seconds since midnight, 1st of January 1970 using a signed, 32-bit integer.

Often called `utime` or `time_t` the following time range can be represented:

```
$ date -d @-2147483647 -Iseconds
1901-12-13T20:45:53+00:00
$
$ date -d @0 -Iseconds
1970-01-01T00:00:00+00:00
$
$ date -d @2147483648 -Iseconds
2038-01-19T03:14:08+00:00
```

The 32-bit nature of `utime` will result in a Y2K-like rollover in 2038 as above, however most systems are already using 64-bit integers and this will unlikely be a problem in practice (touch wood)

- https://en.wikipedia.org/wiki/Unix_time

tm

This is a C hash (technically a struct) that holds a "broken down" version of `utime`. The original looked similar to:

```
struct tm {
  int tm_sec;   // seconds, range 0 to 59
  int tm_min;   // minutes, range 0 to 59
  int tm_hour;  // hours, range 0 to 23
  int tm_mday;  // day of the month, range 1 to 31
  int tm_mon;   // month, range 0 to 11
  int tm_year;  // The number of years since 1900
  int tm_wday;  // day of the week, range 0 to 6
  int tm_yday;  // day in the year, range 0 to 365
  int tm_isdst; // daylight saving time flag
}
```

Note that the range for `tm_sec` is now variously 0-60 or 0-61 (depending on the standard) to allow for a 1-2sec positive leap second. Note also that `tm_isdst` indicates whether this time has had DST applied to it.

- https://en.cppreference.com/w/c/chrono/tm

libc TZ

Rather than set the system clock to local time, Unix systems have traditionally set their clocks to UTC and then calculated the local time by applying a static offset.

Functions to perform these calculations were added to many implementations of the C library `libc`, however this wasn't formally standardised until ANSI C in the late 70s and later POSIX in the mid 80s.

Whether or not to apply DST offset is now usually managed with the timezone database files as discussed below, but most variants of `libc` still retain the ability to read and understand a fairly complex TZ format, which grew up over time before the database approach was widespread.

For these examples we are assuming the Daylight Savings Rules for Victoria that are in use after 2008, which are:

- Daylight Savings starts on the first Sunday of October and runs 01:59:59AEST -> 03:00:00AEDT
- Daylight Savings finishes on the first Sunday of April and runs 02:59:59AEDT -> 02:00:00AEST

In this case, the value of the TZ environment variable can be used to indicate when to apply DST offset:

```
TZ=AEST-10
TZ=AEST-10AEDT
TZ=AEST-10AEDT-11,M10.1.0/2,M4.1.0/3
```

1. Australian Eastern Standard Timezone, UTC+10h, no DST specified (note the sign of offset)
2. added offset for Australian Eastern Daylight Time, but DST changes follow broken Northern Hemisphere dates
3. added definition for DST changes for 2025 to match rule above

For example, going into DST:

```
$ export TZ=AEST-10AEDT-11,M10.1.0/2,M4.1.0/3
$
$ date -Iseconds -d 2025-10-05T01:00:00
2025-10-05T01:00:00+10:00
$ date -Iseconds -d 2025-10-05T02:00:00
date: invalid date ‘2025-10-05T02:00:00’
$ date -Iseconds -d 2025-10-05T03:00:00
2025-10-05T03:00:00+11:00
$
$ date -Iseconds -d 2025-10-05T01:59:59+10:00
2025-10-05T01:59:59+10:00
$ date -Iseconds -d 2025-10-05T02:00:00+10:00
2025-10-05T03:00:00+11:00
$ date -Iseconds -d 2025-10-05T03:00:00+11:00
2025-10-05T03:00:00+11:00
```

and coming out of DST:

```
$ export TZ=AEST-10AEDT-11,M10.1.0/2,M4.1.0/3
$
$ date -Iseconds -d 2025-04-06T01:59:59
2025-04-06T01:59:59+11:00
$ date -Iseconds -d 2025-04-06T02:00:00
2025-04-06T02:00:00+10:00
$
$ date -Iseconds -d 2025-04-06T02:00:00+11:00
2025-04-06T02:00:00+11:00
$ date -Iseconds -d 2025-04-06T02:59:59+11:00
2025-04-06T02:59:59+11:00
$ date -Iseconds -d 2025-04-06T03:00:00+11:00
2025-04-06T02:00:00+10:00
$ date -Iseconds -d 2025-04-06T03:00:00+10:00
2025-04-06T03:00:00+10:00
```

Note that in this example, when converting dates without offset around the changeover into DST, `libc` can't determine which 02:00am we are talking about (with or without DST) and throws an error.

The second set of dates include the non-DST offset for 02:00am and we get the correct time and offset returned, however times within the skipped wallclock time behave oddly:

```
$ export TZ=AEST-10AEDT-11,M10.1.0/2,M4.1.0/3
$
$ date -Iseconds -d 2025-10-05T01:59:59+10:00
2025-10-05T01:59:59+10:00
$ date -Iseconds -d 2025-10-05T02:00:00+11:00
2025-10-05T01:00:00+10:00
$ date -Iseconds -d 2025-10-05T03:00:00+11:00
2025-10-05T03:00:00+11:00
```

The actual dates for changing to and from DST above are only guaranteed to be correct for dates after 2008. If we had wanted to map dates for (eg) 2007, then the end of Daylight Savings was early:

- Daylight Savings starts on the first Sunday of October and runs 01:59:59AEST -> 03:00:00AEDT
- Daylight Savings finishes on the last Sunday of March and runs 02:59:59AEDT -> 02:00:00AEST

This would impact our TZ variable as follows:

```
$ export TZ=AEST-10AEDT-11,M10.1.0/2,M3.5.0/3
$
$ date -Iseconds -d 2025-03-30T01:59:59
2025-03-30T01:59:59+11:00
$ date -Iseconds -d 2025-03-30T02:00:00
2025-03-30T02:00:00+10:00
$
$ date -Iseconds -d 2025-03-30T02:00:00+11:00
2025-03-30T02:00:00+11:00
$ date -Iseconds -d 2025-03-30T02:59:59+11:00
2025-03-30T02:59:59+11:00
$ date -Iseconds -d 2025-03-30T03:00:00+11:00
2025-03-30T02:00:00+10:00
$ date -Iseconds -d 2025-03-30T03:00:00+10:00
2025-03-30T03:00:00+10:00
```

For reference, had we kept the previous value of TZ (valid for dates from 2008) the last set of commands would return the following:

```
$ date -Iseconds -d 2025-03-30T02:00:00+11:00
2025-03-30T02:00:00+11:00
$ date -Iseconds -d 2025-03-30T02:59:59+11:00
2025-03-30T02:59:59+11:00
$ date -Iseconds -d 2025-03-30T03:00:00+11:00
2025-03-30T03:00:00+11:00
$ date -Iseconds -d 2025-03-30T03:00:00+10:00
2025-03-30T04:00:00+11:00
```

Olson/tzdata/zoneinfo database

Clearly having to modify individual system TZ settings by hand to manage Daylight Savings is not workable in practice.

From early 1984 Arthur David Olson began writing code to automate the production of binary tables of timezone offsets and Daylight Savings changes, in collaboration with a large number of contributors. Paul Eggert has been the maintainer of the database since 2005.

Because Daylight Savings is an inherently political problem, dates for Daylight Savings changes are typically gazetted by governments. Before adoption, these announcements need to be distilled and clarified before they can be used. To keep everyone current, the database is usually updated and released every few months to capture upcoming changes.

- https://en.wikipedia.org/wiki/Tz_database

- https://data.iana.org/time-zones/tz-link.html

Some countries are notorious for their disregard for the lead time required to implement and test changes. This was true of Turkey in 2015 whose government opted to keep Daylight Savings an extra two weeks to assist with their upcoming election. Unfortunately this wasn't well publicised and the IATA community were caught somewhat flat-footed. In the end some downstream projects did not get updated by the changeover date and there was a moderate amount of chaos involved.

- https://codeofmatt.com/on-the-timing-of-time-zone-changes/

Partly due to geography and partly due to history, Australia has had a complex relationship with timezones and with Daylight Savings - in 2010 the Australian Parliament produced a report underlining the complexity

- http://www.bom.gov.au/climate/averages/tables/dst_times.shtml
- https://parlinfo.aph.gov.au/parlInfo/download/library/prspub/359V6/upload_binary/359v60.pdf

The files collated in this database use a very simple text format to describe time representation offsets and discontinuities. In many cases the underlying evidence in support of the change is kept within the source file comments and make for interesting reading themselves. For example, this discussion on "double daylight savings" applied by Britain during World War II:

```
:
:
# Date: 4 Jan 89 08:57:25 GMT (Wed)
# From: Jonathan Leffler
# [British Summer Time] is fixed annually by Act of Parliament.
# If you can predict what Parliament will do, you should be in
# politics making a fortune, not computing.

# From Chris Carrier (1996-06-14):
# I remember reading in various wartime issues of the London Times the
# acronym BDST for British Double Summer Time.  Look for the published
# time of sunrise and sunset in The Times, when BDST was in effect, and
# if you find a zone reference it will say, "All times B.D.S.T."

# From Joseph S. Myers (1999-09-02):
# ... some military cables (WO 219/4100 - this is a copy from the
# main SHAEF archives held in the US National Archives, SHAEF/5252/8/516)
# agree that the usage is BDST (this appears in a message dated 17 Feb 1945).

# From Joseph S. Myers (2000-10-03):
# On 18th April 1941, Sir Stephen Tallents of the BBC wrote to Sir
# Alexander Maxwell of the Home Office asking whether there was any
# official designation; the reply of the 21st was that there wasn't
# but he couldn't think of anything better than the "Double British
# Summer Time" that the BBC had been using informally.
# https://www.polyomino.org.uk/british-time/bbc-19410418.png
# https://www.polyomino.org.uk/british-time/ho-19410421.png

# From Sir Alexander Maxwell in the above-mentioned letter (1941-04-21):
# [N]o official designation has as far as I know been adopted for the time
# which is to be introduced in May....
# I cannot think of anything better than "Double British Summer Time"
# which could not be said to run counter to any official description.

# From Paul Eggert (2000-10-02):
# Howse writes (p 157) 'DBST' too, but 'BDST' seems to have been common
# and follows the more usual convention of putting the location name first,
# so we use 'BDST'.
:
:
```

- https://github.com/eggert/tz/blob/main/europe

Unfortunately this trend for keeping supporting information in the comments resulted in a copyright lawsuit that led to the database being pulled offline for a while in 2011 until things were eventually settled.

- https://www.eff.org/cases/astrolabe-v-olson
- https://www.computerworld.com/article/1548822/astrolabe-withdraws-copyright-suit-over-internet-time-zone-database.html

implementation

On Linux systems, compiled binary timezone information is found within `/usr/share/zoneinfo` in files covering definitions by general location. The nominal "authoritative" city for the region is usually provided as the file and symbolic links used to direct other named variant uses as appropriate. For example, here are the current file and links for Australia:

```
/usr/share/zoneinfo/Australia/ACT -> Sydney
/usr/share/zoneinfo/Australia/Adelaide
/usr/share/zoneinfo/Australia/Brisbane
/usr/share/zoneinfo/Australia/Broken_Hill
/usr/share/zoneinfo/Australia/Canberra -> Sydney
/usr/share/zoneinfo/Australia/Currie -> Hobart
/usr/share/zoneinfo/Australia/Darwin
/usr/share/zoneinfo/Australia/Eucla
/usr/share/zoneinfo/Australia/Hobart
/usr/share/zoneinfo/Australia/LHI -> Lord_Howe
/usr/share/zoneinfo/Australia/Lindeman
/usr/share/zoneinfo/Australia/Lord_Howe
/usr/share/zoneinfo/Australia/Melbourne
/usr/share/zoneinfo/Australia/NSW -> Sydney
/usr/share/zoneinfo/Australia/North -> Darwin
/usr/share/zoneinfo/Australia/Perth
/usr/share/zoneinfo/Australia/Queensland -> Brisbane
/usr/share/zoneinfo/Australia/South -> Adelaide
/usr/share/zoneinfo/Australia/Sydney
/usr/share/zoneinfo/Australia/Tasmania -> Hobart
/usr/share/zoneinfo/Australia/Victoria -> Melbourne
/usr/share/zoneinfo/Australia/West -> Perth
/usr/share/zoneinfo/Australia/Yancowinna -> Broken_Hill
```

The timezone that the system uses is found either as the content or symlink target of `/etc/localtime`, for example:

```
/etc/localtime -> /usr/share/zoneinfo/Australia/Melbourne
```

The `zic` and `zdump` tools are included with most OS variants, or compile the versions from the `tzcode` bundle:

- https://www.iana.org/time-zones
- https://data.iana.org/time-zones/releases/tzdata2025b.tar.gz
- https://data.iana.org/time-zones/releases/tzcode2025b.tar.gz

dump DST changes

To find the discontinuities in any known timezone, you can use the `zdump` command. This takes the unrolled binary timezone file and enumerates the entries. For example:

```
$ zdump -V -c 2024,2026 Australia/Melbourne
Australia/Melbourne  Sat Apr  6 15:59:59 2024 UT = Sun Apr  7 02:59:59 2024 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  6 16:00:00 2024 UT = Sun Apr  7 02:00:00 2024 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  5 15:59:59 2024 UT = Sun Oct  6 01:59:59 2024 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  5 16:00:00 2024 UT = Sun Oct  6 03:00:00 2024 AEDT isdst=1 gmtoff=39600
```

```
Australia/Melbourne  Sat Apr  5 15:59:59 2025 UT = Sun Apr  6 02:59:59 2025 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  5 16:00:00 2025 UT = Sun Apr  6 02:00:00 2025 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 15:59:59 2025 UT = Sun Oct  5 01:59:59 2025 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 16:00:00 2025 UT = Sun Oct  5 03:00:00 2025 AEDT isdst=1 gmtoff=39600
```

Some variants of `zdump` don't include the -V or -c options - you can do similar with this command:

```
$ zdump -v Australia/Melbourne |
> sed -ne '/2024 UT/,/2026 UT/p'
Australia/Melbourne  Sat Apr  6 15:59:59 2024 UT = Sun Apr  7 02:59:59 2024 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  6 16:00:00 2024 UT = Sun Apr  7 02:00:00 2024 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  5 15:59:59 2024 UT = Sun Oct  6 01:59:59 2024 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  5 16:00:00 2024 UT = Sun Oct  6 03:00:00 2024 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  5 15:59:59 2025 UT = Sun Apr  6 02:59:59 2025 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  5 16:00:00 2025 UT = Sun Apr  6 02:00:00 2025 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 15:59:59 2025 UT = Sun Oct  5 01:59:59 2025 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 16:00:00 2025 UT = Sun Oct  5 03:00:00 2025 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  4 15:59:59 2026 UT = Sun Apr  5 02:59:59 2026 AEDT isdst=1 gmtoff=39600
```

converting between timezones

Finding the correct time in another timezone is as simple as:

```
$ env TZ=America/Toronto date -Iseconds -d 2025-08-01T09:00:00
2025-08-01T09:00:00-04:00
$ env TZ=Pacific/Auckland date -Iseconds -d 2025-08-01T09:00:00-04:00
2025-08-02T01:00:00+12:00
```

tzdata file format

For example, this is the initial timezone file for Australia contributed by Keith Edmunds in 1986:

```
# Australian Data (for states with DST), standard rules

# Rule  NAME    FROM    TO  TYPE    IN  ON  AT  SAVE    LETTER/S
Rule    Aus 1971    2037    -   Oct lastSun 2:00    1:00    -
Rule    Aus 1972    only    -   Feb 27  3:00    0   -
Rule    Aus 1973    2037    -   Mar Sun>=1  3:00    0   -

#Australian Data, Vic (and NSW except for a variation in 83? that I[kre] forget)

Rule    Aus-Vic 1971    2037    -   Oct lastSun 2:00    1:00    -
Rule    Aus-Vic 1972    only    -   Feb 27  3:00    0   -
Rule    Aus-Vic 1973    1985    -   Mar Sun>=1  3:00    0   -
# is this really forever, or just 86??
Rule    Aus-Vic 1986    2037    -   Mar Sun<=21 3:00    0   -

# Australia - something of a turmoil here
# Zone  NAME        GMTOFF  RULES   FORMAT
Zone    EST     10:00   Aus-Vic EST # rule change, 1986
Zone    Tasmania    10:00   Aus EST # still the standard rules?
Zone    Queensland  10:00   -   EST # Queensland - no DST
Zone    CST     9:30    Aus CST # still the standard rules?
Zone    North       9:30    -   CST # Northern Territory - no DST
Zone    WST     8:00    -   WST # No DST ever, this is simple...
```

- https://github.com/eggert/tz

- https://github.com/eggert/tz/blob/main/australasia

creating a timezone file

Apart from describing the normal wall-clock time for various geographical regions, there are a few niche moments where having a "timezone" that you can easily move dates and times between it and the "real world" can be very very useful.

When any activity has a nominal "day" that crosses local calendar midnight there will be confusion when referring to things that occur near midnight because we need to do extra work to determine which calendar date that belongs to.

Airlines normally resolve this by always referring to their flights both in UTC (which has no DST to apply) and airport localtime (most of which do observe DST) which is an established practice that everyone understands and works because it's more-or-less universal.

Unfortunately for other applications there may still be a need to collate all the content for a "day" and somehow ignore the calendar dates that this spans.

An example of this are train timetables: services might start at (eg) localtime 04:00 and extend to 02:00 the following day - in this case, a timezone with a "midnight" set to wall-clock time 03:00 would help to keep all the services for that day on the same date as they would then fall between 01:00 and 23:00 in this nominal "train timetable" zone.

a worked example

The example we will follow here is that for Melbourne community radio station 3RRR which has a similar problem. They broadcast continuously throughout the year and in this case, their "broadcast clock" starts each day at 06:00am[1].

[1] their "broadcast" week also starts 06:00 on Monday however for this example we're going to stick to just the times

- file:doc/RRR206-The-Trip-Summer-2024-GRID.pdf
- https://www.rrr.org.au/explore/schedule

Here, any programs that broadcast prior to 06:00 on any day are deemed to be part of the previous day. For example their program schedule for Saturday 2025-04-26 looks like the following in the Australia/Melbourne timezone (UTC+10h offset):

```
2025-04-26T06:00:00+10:00 Vital Bits
2025-04-26T09:00:00+10:00 Off The Record
2025-04-26T12:00:00+10:00 Press Colour
2025-04-26T14:00:00+10:00 Twang
2025-04-26T16:00:00+10:00 Stolen Moments
2025-04-26T18:00:00+10:00 Beat Orgy
2025-04-26T20:00:00+10:00 Velvet Haze
2025-04-26T22:00:00+10:00 Livewire
2025-04-27T00:00:00+10:00 The Party Show
2025-04-27T02:00:00+10:00 The Graveyard Shift
2025-04-27T04:00:00+10:00 The Graveyard Shift
2025-04-27T06:00:00+10:00 Vital Bits
```

After mapping into our nominal timezone, the schedule should look like this (now with a UTC+4h offset):

```
2025-04-26T00:00:00+04:00 Vital Bits
2025-04-26T03:00:00+04:00 Off The Record
2025-04-26T06:00:00+04:00 Press Colour
2025-04-26T08:00:00+04:00 Twang
2025-04-26T10:00:00+04:00 Stolen Moments
2025-04-26T12:00:00+04:00 Beat Orgy
2025-04-26T14:00:00+04:00 Velvet Haze
2025-04-26T16:00:00+04:00 Livewire
2025-04-26T18:00:00+04:00 The Party Show
2025-04-26T20:00:00+04:00 The Graveyard Shift
```

```
2025-04-26T22:00:00+04:00 The Graveyard Shift
2025-04-27T00:00:00+04:00 Vital Bits
```

We can confirm that these long-form iso8601 dates map to the same `utime` using `date`:

```
$ date -d '2025-04-26T06:00:00+10:00' +%s
1745611200
$ date -d '2025-04-26T00:00:00+04:00' +%s
1745611200
```

We'll start with a simple zoneinfo file for Victoria where 3RRR is based:

```
# simplistic timezone rules for Victoria - valid from 2008 only

# Rule  name    from    to  type    in  on  at  save    letters
Rule    Vic 2008    max -   October Sun>=1  2:00    1:00    S
Rule    Vic 2008    max -   April   Sun>=1  3:00    0:00    W

# Zone  name    stdoff  rules   format
Zone    Vic 10:00   Vic Vic%s
```

For this zone definition, daylight savings starts at 02:00 on the first Sunday of October and ends at 03:00 on the first Sunday of April.

Rather than stick with slightly confusing `AEDT` / `AEST` I've opted to confuse things further by using short names `VicW` for non-DST "Winter" time and `VicS` for DST "Summer" time

When `zic` runs over this file it will assume that it can write output directly into `/usr/share/zoneinfo/Vic` which means we'll need to use sudo:

```
$ zic etc/vic.zi
zic: Can't remove /usr/share/zoneinfo/Vic: Permission denied
$ sudo zic etc/vic.zi
[sudo] password for mjch:
```

Confirm that it's nominally correct:

```
$ env TZ=Australia/Melbourne date ; env TZ=Vic date
Sun Apr 27 13:54:49 AEST 2025
Sun Apr 27 13:54:49 VicW 2025
$ env TZ=Australia/Melbourne date -Iseconds ; env TZ=Vic date -Iseconds
2025-04-27T13:54:39+10:00
2025-04-27T13:54:39+10:00
```

We can also confirm using `zdump`:

```
$ zdump -V -c 2024,2026 Australia/Melbourne Vic |
> sort -k1.22
Australia/Melbourne  Sat Apr  5 15:59:59 2025 UT = Sun Apr  6 02:59:59 2025 AEDT isdst=1 gmtoff=39600
Vic                  Sat Apr  5 15:59:59 2025 UT = Sun Apr  6 02:59:59 2025 VicS isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  5 16:00:00 2025 UT = Sun Apr  6 02:00:00 2025 AEST isdst=0 gmtoff=36000
Vic                  Sat Apr  5 16:00:00 2025 UT = Sun Apr  6 02:00:00 2025 VicW isdst=0 gmtoff=36000
Australia/Melbourne  Sat Apr  6 15:59:59 2024 UT = Sun Apr  7 02:59:59 2024 AEDT isdst=1 gmtoff=39600
Vic                  Sat Apr  6 15:59:59 2024 UT = Sun Apr  7 02:59:59 2024 VicS isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  6 16:00:00 2024 UT = Sun Apr  7 02:00:00 2024 AEST isdst=0 gmtoff=36000
Vic                  Sat Apr  6 16:00:00 2024 UT = Sun Apr  7 02:00:00 2024 VicW isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 15:59:59 2025 UT = Sun Oct  5 01:59:59 2025 AEST isdst=0 gmtoff=36000
```

```
Vic                  Sat Oct  4 15:59:59 2025 UT = Sun Oct  5 01:59:59 2025 VicW isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 16:00:00 2025 UT = Sun Oct  5 03:00:00 2025 AEDT isdst=1 gmtoff=39600
Vic                  Sat Oct  4 16:00:00 2025 UT = Sun Oct  5 03:00:00 2025 VicS isdst=1 gmtoff=39600
Australia/Melbourne  Sat Oct  5 15:59:59 2024 UT = Sun Oct  6 01:59:59 2024 AEST isdst=0 gmtoff=36000
Vic                  Sat Oct  5 15:59:59 2024 UT = Sun Oct  6 01:59:59 2024 VicW isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  5 16:00:00 2024 UT = Sun Oct  6 03:00:00 2024 AEDT isdst=1 gmtoff=39600
Vic                  Sat Oct  5 16:00:00 2024 UT = Sun Oct  6 03:00:00 2024 VicS isdst=1 gmtoff=39600
```

Note: This "sort" is sorting ASCIIbetically on each line from `Sat` which really only helps to put entries for the same UTC time together for easier eyeball matching

Our initial hand check earlier identified that the timezone offset we need is really just UTC+4h to our Victorian UTC+10h offset. Putting this into our zoneinfo definition:

```
# simplistic timezone rules for 3RRR - valid from 2008 only

# Rule  name    from    to  type    in  on  at  save    letters
Rule    RRR 2008    max -   October Sun>=1  2:00    1:00    S
Rule    RRR 2008    max -   April   Sun>=1  3:00    0:00    W

# Zone  name    stdoff  rules   format
Zone    RRR 4:00    RRR RRR%s
```

... compile and install, then check:

```
$ sudo zic etc/rrr.zi
$ zdump -V -c 2024,2026 Australia/Melbourne RRR |
> sort -k1.22
Australia/Melbourne  Sat Apr  5 15:59:59 2025 UT = Sun Apr  6 02:59:59 2025 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  5 16:00:00 2025 UT = Sun Apr  6 02:00:00 2025 AEST isdst=0 gmtoff=36000
RRR                  Sat Apr  5 21:59:59 2025 UT = Sun Apr  6 02:59:59 2025 RRRS isdst=1 gmtoff=18000
RRR                  Sat Apr  5 22:00:00 2025 UT = Sun Apr  6 02:00:00 2025 RRRW isdst=0 gmtoff=14400
Australia/Melbourne  Sat Apr  6 15:59:59 2024 UT = Sun Apr  7 02:59:59 2024 AEDT isdst=1 gmtoff=39600
Australia/Melbourne  Sat Apr  6 16:00:00 2024 UT = Sun Apr  7 02:00:00 2024 AEST isdst=0 gmtoff=36000
RRR                  Sat Apr  6 21:59:59 2024 UT = Sun Apr  7 02:59:59 2024 RRRS isdst=1 gmtoff=18000
RRR                  Sat Apr  6 22:00:00 2024 UT = Sun Apr  7 02:00:00 2024 RRRW isdst=0 gmtoff=14400
Australia/Melbourne  Sat Oct  4 15:59:59 2025 UT = Sun Oct  5 01:59:59 2025 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  4 16:00:00 2025 UT = Sun Oct  5 03:00:00 2025 AEDT isdst=1 gmtoff=39600
RRR                  Sat Oct  4 21:59:59 2025 UT = Sun Oct  5 01:59:59 2025 RRRW isdst=0 gmtoff=14400
RRR                  Sat Oct  4 22:00:00 2025 UT = Sun Oct  5 03:00:00 2025 RRRS isdst=1 gmtoff=18000
Australia/Melbourne  Sat Oct  5 15:59:59 2024 UT = Sun Oct  6 01:59:59 2024 AEST isdst=0 gmtoff=36000
Australia/Melbourne  Sat Oct  5 16:00:00 2024 UT = Sun Oct  6 03:00:00 2024 AEDT isdst=1 gmtoff=39600
RRR                  Sat Oct  5 21:59:59 2024 UT = Sun Oct  6 01:59:59 2024 RRRW isdst=0 gmtoff=14400
RRR                  Sat Oct  5 22:00:00 2024 UT = Sun Oct  6 03:00:00 2024 RRRS isdst=1 gmtoff=18000
```

After this change, the UTC times don't match and things no longer line up. We need to make the same offset adjustment to the time when DST changes. In our initial zone definition, these were 02:00 and 03:00 but `zic` knows how to deal with negative change times, so:

```
# simplistic timezone rules for 3RRR - valid from 2008 only

# Rule  name    from    to  type    in  on  at  save    letters
Rule    RRR 2008    max -   October Sun>=1  -4:00   1:00    S
Rule    RRR 2008    max -   April   Sun>=1  -3:00   0:00    W

# Zone  name    stdoff  rules   format
Zone    RRR 4:00    RRR RRR%s
```

Compile and test:

```
$ zdump -V -c 2024,2026 Australia/Melbourne RRR | sort -k1.22
RRR                  Sat Apr  5 15:59:59 2025 UT = Sat Apr  5 20:59:59 2025 RRRS isdst=1 gmtoff=18000
Australia/Melbourne  Sat Apr  5 15:59:59 2025 UT = Sun Apr  6 02:59:59 2025 AEDT isdst=1 gmtoff=39600
RRR                  Sat Apr  5 16:00:00 2025 UT = Sat Apr  5 20:00:00 2025 RRRW isdst=0 gmtoff=14400
Australia/Melbourne  Sat Apr  5 16:00:00 2025 UT = Sun Apr  6 02:00:00 2025 AEST isdst=0 gmtoff=36000
RRR                  Sat Apr  6 15:59:59 2024 UT = Sat Apr  6 20:59:59 2024 RRRS isdst=1 gmtoff=18000
Australia/Melbourne  Sat Apr  6 15:59:59 2024 UT = Sun Apr  7 02:59:59 2024 AEDT isdst=1 gmtoff=39600
RRR                  Sat Apr  6 16:00:00 2024 UT = Sat Apr  6 20:00:00 2024 RRRW isdst=0 gmtoff=14400
Australia/Melbourne  Sat Apr  6 16:00:00 2024 UT = Sun Apr  7 02:00:00 2024 AEST isdst=0 gmtoff=36000
RRR                  Sat Oct  4 15:59:59 2025 UT = Sat Oct  4 19:59:59 2025 RRRW isdst=0 gmtoff=14400
Australia/Melbourne  Sat Oct  4 15:59:59 2025 UT = Sun Oct  5 01:59:59 2025 AEST isdst=0 gmtoff=36000
RRR                  Sat Oct  4 16:00:00 2025 UT = Sat Oct  4 21:00:00 2025 RRRS isdst=1 gmtoff=18000
Australia/Melbourne  Sat Oct  4 16:00:00 2025 UT = Sun Oct  5 03:00:00 2025 AEDT isdst=1 gmtoff=39600
RRR                  Sat Oct  5 15:59:59 2024 UT = Sat Oct  5 19:59:59 2024 RRRW isdst=0 gmtoff=14400
Australia/Melbourne  Sat Oct  5 15:59:59 2024 UT = Sun Oct  6 01:59:59 2024 AEST isdst=0 gmtoff=36000
RRR                  Sat Oct  5 16:00:00 2024 UT = Sat Oct  5 21:00:00 2024 RRRS isdst=1 gmtoff=18000
Australia/Melbourne  Sat Oct  5 16:00:00 2024 UT = Sun Oct  6 03:00:00 2024 AEDT isdst=1 gmtoff=39600
```

Now things match and we can convert between an "RRR" time and a wallclock time for Melbourne with the same Daylight Savings change dates.

fine-tuning the DST changeover time

Unfortunately, this isn't quite the end of the story - we still have to adjust the DST changeover time to match RRR practice and avoid impacting two programs `The Party Show` which concludes Sunday morning at 02:00 and `The Graveyard Shift` that runs from 02:00 to 06:00 most days of the week.

Given the break between shows and the Daylight Savings time change coincide, this has the effect of invalidating the endpoint for the show heading into Daylight Savings if we try to rely on our timezone to Do The Right Thing - we can of course put in an absolute offset from UTC to fix this, but that's not the point:

```
$ env TZ=Australia/Melbourne date -d '2024-10-06T02:00:00'
date: invalid date ‘2024-10-06T02:00:00’
$ env TZ=RRR date -d '2024-10-05T20:00:00'
date: invalid date ‘2024-10-05T20:00:00’
```

While `The Party Show` is a regular 2hour show, `The Graveyard Shift` is mainly used for training purposes. Rather than interrupt regular programming, 3RRR choose to absorb the wayward hour within `The Graveyard Shift` making it 3 or 5 hours as needed since the impact will be reduced.

By shifting the DST change back one second we achieve the result we're looking for:

```
# simplistic timezone rules for 3RRR - valid from 2008 only

# Rule  name    from    to  type    in  on  at  save    letters
Rule    RRR 2008    max -   October Sun>=1  -3:59:59    1:00    S
Rule    RRR 2008    max -   April   Sun>=1  -2:59:59    0:00    W

# Zone  name    stdoff  rules   format
Zone    RRR 4:00    RRR RRR%s
```

We can use `faketime` to confirm the behaviour at the changover:

```
$ faketime '2024-10-06 01:59:40+1000' bash
$ while sleep 1 ; do
> echo "$(env TZ=Australia/Melbourne date -Iseconds) - $(env TZ=RRR date -Iseconds)"
```

```
> done
:
:
2024-10-06T01:59:57+10:00 - 2024-10-05T19:59:57+04:00
2024-10-06T01:59:58+10:00 - 2024-10-05T19:59:58+04:00
2024-10-06T01:59:59+10:00 - 2024-10-05T19:59:59+04:00
2024-10-06T03:00:00+11:00 - 2024-10-05T20:00:00+04:00
2024-10-06T03:00:01+11:00 - 2024-10-05T21:00:01+05:00
2024-10-06T03:00:02+11:00 - 2024-10-05T21:00:02+05:00
2024-10-06T03:00:03+11:00 - 2024-10-05T21:00:03+05:00
:
:
```

and same again coming out of daylight savings:

```
$ faketime '2025-04-06 02:59:40+1100' bash
$ while sleep 1 ; do
> echo "$(env TZ=Australia/Melbourne date -Iseconds) - $(env TZ=RRR date -Iseconds)"
> done
:
:
2025-04-06T02:59:57+11:00 - 2025-04-05T20:59:57+05:00
2025-04-06T02:59:58+11:00 - 2025-04-05T20:59:58+05:00
2025-04-06T02:59:59+11:00 - 2025-04-05T20:59:59+05:00
2025-04-06T02:00:00+10:00 - 2025-04-05T20:00:00+04:00
2025-04-06T02:00:01+10:00 - 2025-04-05T20:00:01+04:00
2025-04-06T02:00:02+10:00 - 2025-04-05T20:00:02+04:00
2025-04-06T02:00:03+10:00 - 2025-04-05T20:00:03+04:00
:
:
```

So now `The Party Show` always runs from 18:00:00 to 20:00:00 in the RRR timezone and is always 2 hours long:

```
$ # The Party Show - week prior to DST change
$ expr $(date -d '2024-09-29T20:00:00' +%s) - $(date -d '2024-09-29T18:00:00' +%s)
7200
$ # The Graveyard Shift
$ expr $(date -d '2024-09-30T00:00:00' +%s) - $(date -d '2024-09-29T20:00:00' +%s)
14400
:
:
$ # The Party Show - night of DST change
$ expr $(date -d '2024-10-05T20:00:00' +%s) - $(date -d '2024-10-05T18:00:00' +%s)
7200
$ # The Graveyard Shift - loses 1hour due to change
$ expr $(date -d '2024-10-06T00:00:00' +%s) - $(date -d '2024-10-05T20:00:00' +%s)
10800
:
:
$ # The Party Show - week after DST change
$ expr $(date -d '2024-10-13T20:00:00' +%s) - $(date -d '2024-10-13T18:00:00' +%s)
7200
$ # The Graveyard Shift
$ expr $(date -d '2024-10-14T00:00:00' +%s) - $(date -d '2024-10-13T20:00:00' +%s)
14400
```

Same when exiting Daylight Savings:

```
$ # The Party Show - week prior to DST change
$ expr $(date -d '2025-03-29T20:00:00' +%s) - $(date -d '2025-03-29T18:00:00' +%s)
7200
$ # The Graveyard Shift
$ expr $(date -d '2025-03-30T00:00:00' +%s) - $(date -d '2025-03-29T20:00:00' +%s)
14400
:
:
$ # The Party Show - night of DST change
$ expr $(date -d '2025-04-05T20:00:00' +%s) - $(date -d '2025-04-05T18:00:00' +%s)
7200
$ # The Graveyard Shift - gains 1hour due to change
$ expr $(date -d '2025-04-06T00:00:00' +%s) - $(date -d '2025-04-05T20:00:00' +%s)
18000
:
:
$ # The Party Show - week after DST change
$ expr $(date -d '2025-04-12T20:00:00' +%s) - $(date -d '2025-04-12T18:00:00' +%s)
7200
$ # The Graveyard Shift
$ expr $(date -d '2025-04-13T00:00:00' +%s) - $(date -d '2025-04-12T20:00:00' +%s)
14400
```

PostgreSQL

The absolute best thing about having your own timezone is getting *exactly* the same behaviour in other tools with no extra work. To demonstrate this, here are the same examples above in a PostgreSQL session:

```
$ psql -A -t
psql (14.17 (Ubuntu 14.17-0ubuntu0.22.04.1))
Type "help" for help.

mjch=> set time zone 'RRR';
SET
mjch=> show time zone;
RRR
:
:
mjch=> -- The Party Show - week prior to DST change
mjch=> select '2024-09-29T20:00:00'::timestamptz - '2024-09-29T18:00:00'::timestamptz;
02:00:00
mjch=> -- The Graveyard Shift
mjch=> select '2024-09-30T00:00:00'::timestamptz - '2024-09-29T20:00:00'::timestamptz;
04:00:00
:
:
mjch=> -- The Party Show - night of DST change
mjch=> select '2024-10-05T20:00:00'::timestamptz - '2024-10-05T18:00:00'::timestamptz;
02:00:00
mjch=> -- The Graveyard Shift - loses 1hour due to change
mjch=> select '2024-10-06T00:00:00'::timestamptz - '2024-10-05T20:00:00'::timestamptz;
03:00:00
:
:
mjch=> -- The Party Show - week after DST change
mjch=> select '2024-10-13T20:00:00'::timestamptz - '2024-10-13T18:00:00'::timestamptz;
02:00:00
```

```
mjch=> -- The Graveyard Shift
mjch=> select '2024-10-14T00:00:00'::timestamptz - '2024-10-13T20:00:00'::timestamptz;
04:00:00
:
:
mjch=> -- The Party Show - week prior to DST change
mjch=> select '2025-03-29T20:00:00'::timestamptz - '2025-03-29T18:00:00'::timestamptz;
02:00:00
mjch=> -- The Graveyard Shift
mjch=> select '2025-03-30T00:00:00'::timestamptz - '2025-03-29T20:00:00'::timestamptz;
04:00:00
:
:
mjch=> -- The Party Show - night of DST change
mjch=> select '2025-04-05T20:00:00'::timestamptz - '2025-04-05T18:00:00'::timestamptz;
02:00:00
mjch=> -- The Graveyard Shift - gains 1hour due to change
mjch=> select '2025-04-06T00:00:00'::timestamptz - '2025-04-05T20:00:00'::timestamptz;
05:00:00
:
:
mjch=> -- The Party Show - week after DST change
mjch=> select '2025-04-12T20:00:00'::timestamptz - '2025-04-12T18:00:00'::timestamptz;
02:00:00
mjch=> -- The Graveyard Shift
mjch=> select '2025-04-13T00:00:00'::timestamptz - '2025-04-12T20:00:00'::timestamptz;
04:00:00
```

questions?

Thanks for your attention.