Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'zshcalsys.1' command

$ man zshcalsys.1

ZSHCALSYS(1) General Commands Manual ZSHCALSYS(1)

NAME

zshcalsys - zsh calendar system

DESCRIPTION

The shell is supplied with a series of functions to replace and enhance the traditional Unix calendar programme, which warns the user of immi? nent or future events, details of which are stored in a text file (typ? ically calendar in the user's home directory). The version provided here includes a mechanism for alerting the user when an event is due. In addition functions age, before and after are provided that can be used in a glob qualifier; they allow files to be selected based on their modification times.

The format of the calendar file and the dates used there in and in the age function are described first, then the functions that can be called to examine and modify the calendar file.

The functions here depend on the availability of the zsh/datetime mod? ule which is usually installed with the shell. The library function strptime() must be available; it is present on most recent operating systems.

FILE AND DATE FORMATS

Calendar File Format

The calendar file is by default ~/calendar. This can be configured by the calendar-file style, see the section STYLES below. The basic for?

mat consists of a series of separate lines, with no indentation, each including a date and time specification followed by a description of the event.

Various enhancements to this format are supported, based on the syntax of Emacs calendar mode. An indented line indicates a continuation line that continues the description of the event from the preceding line (note the date may not be continued in this way). An initial ampersand (&) is ignored for compatibility.

An indented line on which the first non-whitespace character is # is not displayed with the calendar entry, but is still scanned for infor? mation. This can be used to hide information useful to the calendar system but not to the user, such as the unique identifier used by cal? endar_add.

The Emacs extension that a date with no description may refer to a num? ber of succeeding events at different times is not supported.

Unless the done-file style has been altered, any events which have been processed are appended to the file with the same name as the calendar file with the suffix .done, hence ~/calendar.done by default.

An example is shown below.

Date Format

The format of the date and time is designed to allow flexibility with? out admitting ambiguity. (The words `date' and `time' are both used in the documentation below; except where specifically noted this implies a string that may include both a date and a time specification.) Note that there is no localization support; month and day names must be in English and separator characters are fixed. Matching is case insensi? tive, and only the first three letters of the names are significant, although as a special case a form beginning "month" does not match "Monday". Furthermore, time zones are not handled; all times are as? sumed to be local.

It is recommended that, rather than exploring the intricacies of the system, users find a date format that is natural to them and stick to it. This will avoid unexpected effects. Various key facts should be

noted.

? In particular, note the confusion between month/day/year and day/month/year when the month is numeric; these formats should be avoided if at all possible. Many alternatives are available.

? The year must be given in full to avoid confusion, and only years from 1900 to 2099 inclusive are matched.

The following give some obvious examples; users finding here a format they like and not subject to vagaries of style may skip the full de? scription. As dates and times are matched separately (even though the time may be embedded in the date), any date format may be mixed with any format for the time of day provide the separators are clear (white? space, colons, commas).

2007/04/03 13:13

2007/04/03:13:13

2007/04/03 1:13 pm

3rd April 2007, 13:13

April 3rd 2007 1:13 p.m.

Apr 3, 2007 13:13

Tue Apr 03 13:13:00 2007

13:13 2007/apr/3

More detailed rules follow.

Times are parsed and extracted before dates. They must use colons to separate hours and minutes, though a dot is allowed before seconds if they are present. This limits time formats to the following:

? HH:MM[:SS[.FFFFF]] [am|pm|a.m.|p.m.]

? HH:MM.SS[.FFFFF] [am|pm|a.m.|p.m.]

Here, square brackets indicate optional elements, possibly with alter? natives. Fractions of a second are recognised but ignored. For abso? lute times (the normal format require by the calendar file and the age, before and after functions) a date is mandatory but a time of day is not; the time returned is at the start of the date. One variation is allowed: if a.m. or p.m. or one of their variants is present, an hour without a minute is allowed, e.g. 3 p.m..

Time zones are not handled, though if one is matched following a time specification it will be removed to allow a surrounding date to be parsed. This only happens if the format of the timezone is not too un? usual. The following are examples of forms that are understood:

+0100

GMT

GMT-7

CET+1CDT

Any part of the timezone that is not numeric must have exactly three capital letters in the name.

Dates suffer from the ambiguity between DD/MM/YYYY and MM/DD/YYYY. It is recommended this form is avoided with purely numeric dates, but use of ordinals, eg. 3rd/04/2007, will resolve the ambiguity as the ordinal is always parsed as the day of the month. Years must be four digits (and the first two must be 19 or 20); 03/04/08 is not recognised. Other numbers may have leading zeroes, but they are not required. The following are handled:

? YYYY/MM/DD

? YYYY-MM-DD

? YYYY/MNM/DD

? YYYY-MNM-DD

? DD[th|st|rd] MNM[,] [YYYY]

? MNM DD[th|st|rd][,] [YYYY]

? DD[th|st|rd]/MM[,] YYYY

? DD[th|st|rd]/MM/YYYY

? MM/DD[th|st|rd][,] YYYY

? MM/DD[th|st|rd]/YYYY

Here, MNM is at least the first three letters of a month name, matched case-insensitively. The remainder of the month name may appear but its contents are irrelevant, so janissary, febrile, martial, apricot, maybe, junta, etc. are happily handled.

Where the year is shown as optional, the current year is assumed. There are only two such cases, the form Jun 20 or 14 September (the

only two commonly occurring forms, apart from a "the" in some forms of English, which isn't currently supported). Such dates will of course become ambiguous in the future, so should ideally be avoided.

Times may follow dates with a colon, e.g. 1965/07/12:09:45; this is in order to provide a format with no whitespace. A comma and whitespace are allowed, e.g. 1965/07/12, 09:45. Currently the order of these sep? arators is not checked, so illogical formats such as 1965/07/12, : ,09:45 will also be matched. For simplicity such variations are not shown in the list above. Otherwise, a time is only recognised as being associated with a date if there is only whitespace in between, or if the time was embedded in the date.

Days of the week are not normally scanned, but will be ignored if they occur at the start of the date pattern only. However, in contexts where it is useful to specify dates relative to today, days of the week with no other date specification may be given. The day is assumed to be either today or within the past week. Likewise, the words yester? day, today and tomorrow are handled. All matches are case-insensitive. Hence if today is Monday, then Sunday is equivalent to yesterday, Mon? day is equivalent to today, but Tuesday gives a date six days ago. This is not generally useful within the calendar file. Dates in this format may be combined with a time specification; for example Tomorrow, 8 p.m..

For example, the standard date format:

Fri Aug 18 17:00:48 BST 2006

is handled by matching HH:MM:SS and removing it together with the matched (but unused) time zone. This leaves the following:

Fri Aug 18 2006

Fri is ignored and the rest is matched according to the standard rules.

Relative Time Format

In certain places relative times are handled. Here, a date is not al? lowed; instead a combination of various supported periods are allowed, together with an optional time. The periods must be in order from most to least significant.

In some cases, a more accurate calculation is possible when there is an anchor date: offsets of months or years pick the correct day, rather than being rounded, and it is possible to pick a particular day in a month as `(1st Friday)', etc., as described in more detail below.

Anchors are available in the following cases. If one or two times are passed to the function calendar, the start time acts an anchor for the end time when the end time is relative (even if the start time is im? plicit). When examining calendar files, the scheduled event being ex? amined anchors the warning time when it is given explicitly by means of the WARN keyword; likewise, the scheduled event anchors a repetition period when given by the RPT keyword, so that specifications such as RPT 2 months, 3rd Thursday are handled properly. Finally, the -R argu? ment to calendar_scandate directly provides an anchor for relative cal? culations.

The periods handled, with possible abbreviations are:

Years years, yrs, ys, year, yr, y, yearly. A year is 365.25 days un? less there is an anchor.

Months months, mons, mnths, mths, month, mon, mnth, mth, monthly. Note that m, ms, mn, mns are ambiguous and are not handled. A month is a period of 30 days rather than a calendar month unless there is an anchor.

Weeks weeks, wks, ws, week, wk, w, weekly

Days days, dys, ds, day, dy, d, daily

Hours hours, hrs, hs, hour, hr, h, hourly

Minutes

minutes, mins, minute, min, but not m, ms, mn or mns

Seconds

seconds, secs, ss, second, sec, s

Spaces between the numbers are optional, but are required between items, although a comma may be used (with or without spaces).

The forms yearly to hourly allow the number to be omitted; it is as? sumed to be 1. For example, 1 d and daily are equivalent. Note that using those forms with plurals is confusing; 2 yearly is the same as 2

years, not twice yearly, so it is recommended they only be used without numbers.

When an anchor time is present, there is an extension to handle regular events in the form of the nth someday of the month. Such a specifica? tion must occur immediately after any year and month specification, but before any time of day, and must be in the form n(th|st|rd) day, for example 1st Tuesday or 3rd Monday. As in other places, days are matched case insensitively, must be in English, and only the first three letters are significant except that a form beginning `month' does not match `Monday'. No attempt is made to sanitize the resulting date; attempts to squeeze too many occurrences into a month will push the day into the next month (but in the obvious fashion, retaining the correct day of the week).

Here are some examples:

30 years 3 months 4 days 3:42:41

14 days 5 hours

Monthly, 3rd Thursday

4d,10hr

Example

Here is an example calendar file. It uses a consistent date format, as recommended above.

Feb 1, 2006 14:30 Pointless bureaucratic meeting

Mar 27, 2006 11:00 Mutual recrimination and finger pointing

Bring water pistol and waterproofs

Mar 31, 2006 14:00 Very serious managerial pontification

UID 12C7878A9A50

Apr 10, 2006 13:30 Even more pointless blame assignment exercise WARN 30 mins

May 18, 2006 16:00 Regular moaning session RPT monthly, 3rd Thursday

The second entry has a continuation line. The third entry has a con? tinuation line that will not be shown when the entry is displayed, but the unique identifier will be used by the calendar_add function when updating the event. The fourth entry will produce a warning 30 minutes before the event (to allow you to equip yourself appropriately). The

fifth entry repeats after a month on the 3rd Thursday, i.e. June 15, 2006, at the same time.

USER FUNCTIONS

This section describes functions that are designed to be called directly by the user. The first part describes those functions associated with the user's calendar; the second part describes the use in glob qualifiers.

Calendar system functions

calendar [-abdDsv] [-C calfile] [-n num] [-S showprog] [[start] end]

calendar -r [-abdDrsv] [-C calfile] [-n num] [-S showprog] [start]

Show events in the calendar.

With no arguments, show events from the start of today until the end of the next working day after today. In other words, if today is Friday, Saturday, or Sunday, show up to the end of the following Monday, otherwise show today and tomorrow.

If end is given, show events from the start of today up to the time and date given, which is in the format described in the previous section. Note that if this is a date the time is assumed to be midnight at the start of the date, so that effectively this shows all events before the given date.

end may start with a +, in which case the remainder of the specification is a relative time format as described in the previous section indicating the range of time from the start time that is to be included.

If start is also given, show events starting from that time and date. The word now can be used to indicate the current time.

To implement an alert when events are due, include calendar -s in your ~/.zshrc file.

Options:

-a Show all items in the calendar, regardless of the start and end.

-b Brief: don't display continuation lines (i.e. indented lines following the line with the date/time), just the first line.

-B lines

Brief: display at most the first lines lines of the cal? endar entry. `-B 1' is equivalent to `-b'.

-C calfile

Explicitly specify a calendar file instead of the value of the calendar-file style or the default ~/calendar.

-d Move any events that have passed from the calendar file to the "done" file, as given by the done-file style or the default which is the calendar file with .done ap? pended. This option is implied by the -s option.

-D Turns off the option -d, even if the -s option is also present.

-n num, -num

Show at least num events, if present in the calendar file, regardless of the start and end.

-r Show all the remaining options in the calendar, ignoring the given end time. The start time is respected; any ar? gument given is treated as a start time.

-s Use the shell's sched command to schedule a timed event that will warn the user when an event is due. Note that the sched command only runs if the shell is at an inter? active prompt; a foreground task blocks the scheduled task from running until it is finished.

The timed event usually runs the programme calendar_show to show the event, as described in the section UTILITY FUNCTIONS below.

By default, a warning of the event is shown five minutes before it is due. The warning period can be configured by the style warn-time or for a single calendar entry by including WARN reltime in the first line of the entry,

where reltime is one of the usual relative time formats.

A repeated event may be indicated by including RPT rel? date in the first line of the entry. After the scheduled event has been displayed it will be re-entered into the calendar file at a time reldate after the existing event. Note that this is currently the only use made of the re? peat count, so that it is not possible to query the schedule for a recurrence of an event in the calendar un? til the previous event has passed.

If RPT is used, it is also possible to specify that cer? tain recurrences of an event are rescheduled or can? celled. This is done with the OCCURRENCE keyword, fol? lowed by whitespace and the date and time of the occur? rence in the regular sequence, followed by whitespace and either the date and time of the rescheduled event or the exact string CANCELLED. In this case the date and time must be in exactly the "date with local time" format used by the text/calendar MIME type (RFC 2445), <YYYY><MM><DD>T<hh><mm><ss> (note the presence of the literal character T). The first word (the regular recur? rence) may be something other than a proper date/time to indicate that the event is additional to the normal se? quence; a convention that retains the formatting appear? ance is XXXXXXXXTXXXXXX.

Furthermore, it is useful to record the next regular re? currence (as then the displayed date may be for a rescheduled event so cannot be used for calculating the regular sequence). This is specified by RECURRENCE and a time or date in the same format. calendar_add adds such an indication when it encounters a recurring event that does not include one, based on the headline date/time.

If calendar_add is used to update occurrences the UID keyword described there should be present in both the ex?

isting entry and the added occurrence in order to iden?
tify recurring event sequences.
For example,

Thu May 6, 2010 11:00 Informal chat RPT 1 week
RECURRENCE 20100506T110000
OCCURRENCE 20100513T110000 20100513T120000
OCCURRENCE 20100520T110000 CANCELLED

The event that occurs at 11:00 on 13th May 2010 is
rescheduled an hour later. The event that occurs a week
later is cancelled. The occurrences are given on a con?
tinuation line starting with a # character so will not
usually be displayed as part of the event. As elsewhere,
no account of time zones is taken with the times. After
the next event occurs the headline date/time will be `Thu
May 13, 2010 12:00' while the RECURRENCE date/time will
be `20100513T110000' (note that cancelled and moved
events are not taken account of in the RECURRENCE, which
records what the next regular recurrence is, but they are
accounted for in the headline date/time).
It is safe to run calendar -s to reschedule an existing
event (if the calendar file has changed, for example),
and also to have it running in multiples instances of the
shell since the calendar file is locked when in use.
By default, expired events are moved to the "done" file;
see the -d option. Use -D to prevent this.

-S showprog
Explicitly specify a programme to be used for showing
events instead of the value of the show-prog style or the
default calendar_show.

-v Verbose: show more information about stages of process?
ing. This is useful for confirming that the function has
successfully parsed the dates in the calendar file.

Adds a single event to the calendar in the appropriate location. The event can contain multiple lines, as described in the sec? tion Calendar File Format above. Using this function ensures that the calendar file is sorted in date and time order. It also makes special arrangements for locking the file while it is altered. The old calendar is left in a file with the suffix .old.

The option -B indicates that backing up the calendar file will be handled by the caller and should not be performed by calen? dar_add. The option -L indicates that calendar_add does not need to lock the calendar file as it is already locked. These options will not usually be needed by users.

If the style reformat-date is true, the date and time of the new entry will be rewritten into the standard date format: see the descriptions of this style and the style date-format.

The function can use a unique identifier stored with each event to ensure that updates to existing events are treated correctly. The entry should contain the word UID, followed by whitespace, followed by a word consisting entirely of hexadecimal digits of arbitrary length (all digits are significant, including leading zeroes). As the UID is not directly useful to the user, it is convenient to hide it on an indented continuation line starting with a #, for example:

```
Aug 31, 2007 09:30  Celebrate the end of the holidays
 # UID 045B78A0
```

The second line will not be shown by the calendar function.

It is possible to specify the RPT keyword followed by CANCELLED instead of a relative time. This causes any matched event or series of events to be cancelled (the original event does not have to be marked as recurring in order to be cancelled by this method). A UID is required in order to match an existing event in the calendar.

calendar_add will attempt to manage recurrences and occurrences

of repeating events as described for event scheduling by calen?
dar -s above. To reschedule or cancel a single event calen?
dar_add should be called with an entry that includes the correct
UID but does not include the RPT keyword as this is taken to
mean the entry applies to a series of repeating events and hence
replaces all existing information. Each rescheduled or can?
celled occurrence must have an OCCURRENCE keyword in the entry
passed to calendar_add which will be merged into the calendar
file. Any existing reference to the occurrence is replaced. An
occurrence that does not refer to a valid existing event is
added as a one-off occurrence to the same calendar entry.

calendar_edit

This calls the user's editor to edit the calendar file. If
there are arguments, they are taken as the editor to use (the
file name is appended to the commands); otherwise, the editor is
given by the variable VISUAL, if set, else the variable EDITOR.
If the calendar scheduler was running, then after editing the
file calendar -s is called to update it.
This function locks out the calendar system during the edit.
Hence it should be used to edit the calendar file if there is
any possibility of a calendar event occurring meanwhile. Note
this can lead to another shell with calendar functions enabled
hanging waiting for a lock, so it is necessary to quit the edi?
tor as soon as possible.

calendar_parse calendar-entry

This is the internal function that analyses the parts of a cal?
endar entry, which is passed as the only argument. The function
returns status 1 if the argument could not be parsed as a calen?
dar entry and status 2 if the wrong number of arguments were
passed; it also sets the parameter reply to an empty associative
array. Otherwise, it returns status 0 and sets elements of the
associative array reply as follows:
time The time as a string of digits in the same units as

$EPOCHSECONDS

schedtime

The regularly scheduled time. This may differ from the actual event time time if this is a recurring event and the next occurrence has been rescheduled. Then time gives the actual time and schedtime the time of the regu? lar recurrence before modification.

text1 The text from the line not including the date and time of the event, but including any WARN or RPT keywords and values.

warntime

Any warning time given by the WARN keyword as a string of digits containing the time at which to warn in the same units as $EPOCHSECONDS. (Note this is an absolute time, not the relative time passed down.) Not set no WARN key? word and value were matched.

warnstr

The raw string matched after the WARN keyword, else un? set.

rpttime

Any recurrence time given by the RPT keyword as a string of digits containing the time of the recurrence in the same units as $EPOCHSECONDS. (Note this is an absolute time.) Not set if no RPT keyword and value were matched.

schedrpttime

The next regularly scheduled occurrence of a recurring event before modification. This may differ from rpttime, which is the actual time of the event that may have been rescheduled from the regular time.

rptstr The raw string matched after the RPT keyword, else unset.

text2 The text from the line after removal of the date and any keywords and values.

The given date-spec is interpreted and the corresponding date and time printed. If the initial date-spec begins with a + or - it is treated as relative to the current time; date-specs after the first are treated as relative to the date calculated so far and a leading + is optional in that case. This allows one to use the system as a date calculator. For example, calen? dar_showdate '+1 month, 1st Friday' shows the date of the first Friday of next month.

With the option -r nothing is printed but the value of the date and time in seconds since the epoch is stored in the parameter REPLY.

With the option -f fmt the given date/time conversion format is passed to strftime; see notes on the date-format style below.

In order to avoid ambiguity with negative relative date specifi? cations, options must occur in separate words; in other words, -r and -f should not be combined in the same word.

calendar_sort

Sorts the calendar file into date and time order. The old calendar is left in a file with the suffix .old.

Glob qualifiers

age The function age can be autoloaded and use separately from the calendar system, although it uses the function calendar_scandate for date formatting. It requires the zsh/stat builtin, but uses only the builtin zstat.

age selects files having a given modification time for use as a glob qualifier. The format of the date is the same as that un? derstood by the calendar system, described in the section FILE AND DATE FORMATS above.

The function can take one or two arguments, which can be sup? plied either directly as command or arguments, or separately as shell parameters.

```
print *(e:age 2006/10/04 2006/10/09:)
```

The example above matches all files modified between the start

of those dates. The second argument may alternatively be a rel?
ative time introduced by a +:

print *(e:age 2006/10/04 +5d:)

The example above is equivalent to the previous example.
In addition to the special use of days of the week, today and
yesterday, times with no date may be specified; these apply to
today. Obviously such uses become problematic around midnight.

print *(e-age 12:00 13:30-)

The example above shows files modified between 12:00 and 13:00
today.

print *(e:age 2006/10/04:)

The example above matches all files modified on that date. If
the second argument is omitted it is taken to be exactly 24
hours after the first argument (even if the first argument con?
tains a time).

print *(e-age 2006/10/04:10:15 2006/10/04:10:45-)

The example above supplies times. Note that whitespace within
the time and date specification must be quoted to ensure age re?
ceives the correct arguments, hence the use of the additional
colon to separate the date and time.

AGEREF=2006/10/04:10:15

AGEREF2=2006/10/04:10:45

print *(+age)

This shows the same example before using another form of argu?
ment passing. The dates and times in the parameters AGEREF and
AGEREF2 stay in effect until unset, but will be overridden if
any argument is passed as an explicit argument to age. Any ex?
plicit argument causes both parameters to be ignored.
Instead of an explicit date and time, it's possible to use the
modification time of a file as the date and time for either ar?
gument by introducing the file name with a colon:

print *(e-age :file1-)

matches all files created on the same day (24 hours starting

from midnight) as file1.

print *(e-age :file1 :file2-)

matches all files modified no earlier than file1 and no later than file2; precision here is to the nearest second.

after

before The functions after and before are simpler versions of age that take just one argument. The argument is parsed similarly to an argument of age; if it is not given the variable AGEREF is con? sulted. As the names of the functions suggest, a file matches if its modification time is after or before the time and date specified. If a time only is given the date is today.

The two following examples are therefore equivalent:

print *(e-after 12:00-)

print *(e-after today:12:00-)

STYLES

The zsh style mechanism using the zstyle command is describe in zshmod? ules(1). This is the same mechanism used in the completion system.

The styles below are all examined in the context :datetime:function:, for example :datetime:calendar:.

calendar-file

The location of the main calendar. The default is ~/calendar.

date-format

A strftime format string (see strftime(3)) with the zsh exten? sions providing various numbers with no leading zero or space if the number is a single digit as described for the %D{string} prompt format in the section EXPANSION OF PROMPT SEQUENCES in zshmisc(1).

This is used for outputting dates in calendar, both to support the -v option and when adding recurring events back to the cal? endar file, and in calendar_showdate as the final output format.

If the style is not set, the default used is similar the stan? dard system format as output by the date command (also known as `ctime format'): `%a %b %d %H:%M:%S %Z %Y'.

done-file

The location of the file to which events which have passed are appended. The default is the calendar file location with the suffix .done. The style may be set to an empty string in which case a "done" file will not be maintained.

reformat-date

Boolean, used by calendar_add. If it is true, the date and time of new entries added to the calendar will be reformatted to the format given by the style date-format or its default. Only the date and time of the event itself is reformatted; any subsidiary dates and times such as those associated with repeat and warning times are left alone.

show-prog

The programme run by calendar for showing events. It will be passed the start time and stop time of the events requested in seconds since the epoch followed by the event text. Note that calendar -s uses a start time and stop time equal to one another to indicate alerts for specific events.

The default is the function calendar_show.

warn-time

The time before an event at which a warning will be displayed, if the first line of the event does not include the text EVENT reltime. The default is 5 minutes.

UTILITY FUNCTIONS

calendar_lockfiles

Attempt to lock the files given in the argument. To prevent problems with network file locking this is done in an ad hoc fashion by attempting to create a symbolic link to the file with the name file.lockfile. No other system level functions are used for locking, i.e. the file can be accessed and modified by any utility that does not use this mechanism. In particular, the user is not prevented from editing the calendar file at the same time unless calendar_edit is used.

Three attempts are made to lock the file before giving up. If the module zsh/zselect is available, the times of the attempts are jittered so that multiple instances of the calling function are unlikely to retry at the same time.

The files locked are appended to the array lockfiles, which should be local to the caller.

If all files were successfully locked, status zero is returned, else status one.

This function may be used as a general file locking function, although this will only work if only this mechanism is used to lock files.

calendar_read

This is a backend used by various other functions to parse the calendar file, which is passed as the only argument. The array calendar_entries is set to the list of events in the file; no pruning is done except that ampersands are removed from the start of the line. Each entry may contain multiple lines.

calendar_scandate

This is a generic function to parse dates and times that may be used separately from the calendar system. The argument is a date or time specification as described in the section FILE AND DATE FORMATS above. The parameter REPLY is set to the number of seconds since the epoch corresponding to that date or time. By default, the date and time may occur anywhere within the given argument.

Returns status zero if the date and time were successfully parsed, else one.

Options:

-a The date and time are anchored to the start of the argu? ment; they will not be matched if there is preceding text.

-A The date and time are anchored to both the start and end of the argument; they will not be matched if the is any

other text in the argument.

-d Enable additional debugging output.

-m Minus. When -R anchor_time is also given the relative time is calculated backwards from anchor_time.

-r The argument passed is to be parsed as a relative time.

-R anchor_time

The argument passed is to be parsed as a relative time. The time is relative to anchor_time, a time in seconds since the epoch, and the returned value is the absolute time corresponding to advancing anchor_time by the rela? tive time given. This allows lengths of months to be correctly taken into account. If the final day does not exist in the given month, the last day of the final month is given. For example, if the anchor time is during 31st January 2007 and the relative time is 1 month, the final time is the same time of day during 28th February 2007.

-s In addition to setting REPLY, set REPLY2 to the remainder of the argument after the date and time have been stripped. This is empty if the option -A was given.

-t Allow a time with no date specification. The date is as? sumed to be today. The behaviour is unspecified if the iron tongue of midnight is tolling twelve.

calendar_show

The function used by default to display events. It accepts a start time and end time for events, both in epoch seconds, and an event description.

The event is always printed to standard output. If the command line editor is active (which will usually be the case) the com? mand line will be redisplayed after the output.

If the parameter DISPLAY is set and the start and end times are the same (indicating a scheduled event), the function uses the command xmessage to display a window with the event details.

As the system is based entirely on shell functions (with a little sup?
port from the zsh/datetime module) the mechanisms used are not as ro?
bust as those provided by a dedicated calendar utility. Consequently
the user should not rely on the shell for vital alerts.

There is no calendar_delete function.

There is no localization support for dates and times, nor any support
for the use of time zones.

Relative periods of months and years do not take into account the vari?
able number of days.

The calendar_show function is currently hardwired to use xmessage for
displaying alerts on X Window System displays. This should be config?
urable and ideally integrate better with the desktop.

calendar_lockfiles hangs the shell while waiting for a lock on a file.
If called from a scheduled task, it should instead reschedule the event
that caused it.