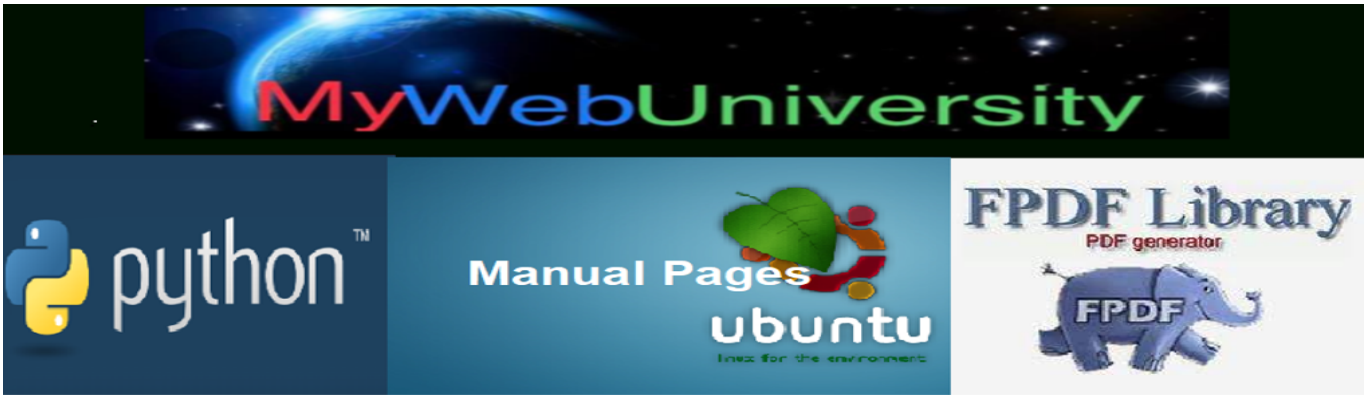

Rocky Enterprise Linux 9.2 Manual Pages on command 'Time::Local.3perl'

$ man Time::Local.3perl

Time::Local(3perl) Perl Programmers Reference Guide Time::Local(3perl)

NAME

Time::Local - Efficiently compute time from local and GMT time

VERSION

version 1.30

SYNOPSIS

```
use Time::Local qw( timelocal_posix timegm_posix );
my $time = timelocal_posix( $sec, $min, $hour, $mday, $mon, $year );
my $time = timegm_posix( $sec, $min, $hour, $mday, $mon, $year );
```

DESCRIPTION

This module provides functions that are the inverse of built-in perl functions "localtime()" and "gmtime()". They accept a date as a six-element array, and return the corresponding time(2) value in seconds since the system epoch (Midnight, January 1, 1970 GMT on Unix, for example). This value can be positive or negative, though POSIX only requires support for positive values, so dates before the system's epoch may not work on all operating systems.

It is worth drawing particular attention to the expected ranges for the values provided. The value for the day of the month is the actual day (i.e. 1..31), while the month is the number of months since January (0..11). This is consistent with the values returned from "localtime()" and "gmtime()".

FUNCTIONS

"timelocal_posix()" and "timegm_posix()"

These functions are the exact inverse of Perl's built-in "localtime" and "gmtime"

functions. That means that calling "timelocal_posix(localtime($value))" will always give you the same $value you started with. The same applies to "timegm_posix(gmtime($value))".

The one exception is when the value returned from "localtime()" represents an ambiguous local time because of a DST change. See the documentation below for more details.

These functions expect the year value to be the number of years since 1900, which is what the "localtime()" and "gmtime()" built-ins returns.

They perform range checking by default on the input $sec, $min, $hour, $mday, and $mon values and will croak (using "Carp::croak()") if given a value outside the allowed ranges.

While it would be nice to make this the default behavior, that would almost certainly break a lot of code, so you must explicitly import these functions and use them instead of the default "timelocal()" and "timegm()".

You are strongly encouraged to use these functions in any new code which uses this module. It will almost certainly make your code's behavior less surprising.

"timelocal_modern()" and "timegm_modern()"

When "Time::Local" was first written, it was a common practice to represent years as a two-digit value like 99 for 1999 or 1 for 2001. This caused all sorts of problems (google "Y2K problem" if you're very young) and developers eventually realized that this was a terrible idea.

The default exports of "timelocal()" and "timegm()" do a complicated calculation when given a year value less than 1000. This leads to surprising results in many cases. See "Year Value Interpretation" for details.

The "time*_modern()" functions do not do this year munging and simply take the year value as provided.

They perform range checking by default on the input $sec, $min, $hour, $mday, and $mon values and will croak (using "Carp::croak()") if given a value outside the allowed ranges.

"timelocal()" and "timegm()"

This module exports two functions by default, "timelocal()" and "timegm()".

They perform range checking by default on the input $sec, $min, $hour, $mday, and $mon values and will croak (using "Carp::croak()") if given a value outside the allowed ranges.

Warning: The year value interpretation that these functions and their nocheck variants use will almost certainly lead to bugs in your code, if not now, then in the future. You are strongly discouraged from using these in new code, and you should convert old code to

using either the *_posix or *_modern functions if possible.

"timelocal_nocheck()" and "timegm_nocheck()"

If you are working with data you know to be valid, you can use the "nocheck" variants, "timelocal_nocheck()" and "timegm_nocheck()". These variants must be explicitly imported.

If you supply data which is not valid (month 27, second 1,000) the results will be unpredictable (so don't do that).

Note that my benchmarks show that this is just a 3% speed increase over the checked versions, so unless calling "Time::Local" is the hottest spot in your application, using these nocheck variants is unlikely to have much impact on your application.

Year Value Interpretation

This does not apply to the *_posix or *_modern functions. Use those exports if you want to ensure consistent behavior as your code ages.

Strictly speaking, the year should be specified in a form consistent with "localtime()", i.e. the offset from 1900. In order to make the interpretation of the year easier for humans, however, who are more accustomed to seeing years as two-digit or four-digit values, the following conventions are followed:

- Years greater than 999 are interpreted as being the actual year, rather than the offset from 1900. Thus, 1964 would indicate the year Martin Luther King won the Nobel prize, not the year 3864.
- Years in the range 100..999 are interpreted as offset from 1900, so that 112 indicates 2012. This rule also applies to years less than zero (but see note below regarding date range).
- Years in the range 0..99 are interpreted as shorthand for years in the rolling "current century," defined as 50 years on either side of the current year. Thus, today, in 1999, 0 would refer to 2000, and 45 to 2045, but 55 would refer to 1955. Twenty years from now, 55 would instead refer to 2055. This is messy, but matches the way people currently think about two digit dates. Whenever possible, use an absolute four digit year instead.

The scheme above allows interpretation of a wide range of dates, particularly if 4-digit years are used. But it also means that the behavior of your code changes as time passes, because the rolling "current century" changes each year.

Limits of time_t

On perl versions older than 5.12.0, the range of dates that can be actually be handled

depends on the size of "time_t" (usually a signed integer) on the given platform.

Currently, this is 32 bits for most systems, yielding an approximate range from Dec 1901 to Jan 2038.

Both "timelocal()" and "timegm()" croak if given dates outside the supported range.

As of version 5.12.0, perl has stopped using the time implementation of the operating system it's running on. Instead, it has its own implementation of those routines with a safe range of at least +/- 2**52 (about 142 million years)

Ambiguous Local Times (DST)

Because of DST changes, there are many time zones where the same local time occurs for two different GMT times on the same day. For example, in the "Europe/Paris" time zone, the local time of 2001-10-28 02:30:00 can represent either 2001-10-28 00:30:00 GMT, or 2001-10-28 01:30:00 GMT.

When given an ambiguous local time, the timelocal() function will always return the epoch for the earlier of the two possible GMT times.

Non-Existent Local Times (DST)

When a DST change causes a locale clock to skip one hour forward, there will be an hour's worth of local times that don't exist. Again, for the "Europe/Paris" time zone, the local clock jumped from 2001-03-25 01:59:59 to 2001-03-25 03:00:00.

If the "timelocal()" function is given a non-existent local time, it will simply return an epoch value for the time one hour later.

Negative Epoch Values

On perl version 5.12.0 and newer, negative epoch values are fully supported.

On older versions of perl, negative epoch ("time_t") values, which are not officially supported by the POSIX standards, are known not to work on some systems. These include MacOS (pre-OSX) and Win32.

On systems which do support negative epoch values, this module should be able to cope with dates before the start of the epoch, down the minimum value of time_t for the system.

IMPLEMENTATION

These routines are quite efficient and yet are always guaranteed to agree with "localtime()" and "gmtime()". We manage this by caching the start times of any months we've seen before. If we know the start time of the month, we can always calculate any time within the month. The start times are calculated using a mathematical formula. Unlike other algorithms that do multiple calls to "gmtime()".

The "timelocal()" function is implemented using the same cache. We just assume that we're translating a GMT time, and then fudge it when we're done for the timezone and daylight savings arguments. Note that the timezone is evaluated for each date because countries occasionally change their official timezones. Assuming that "localtime()" corrects for these changes, this routine will also be correct.

AUTHORS EMERITUS

This module is based on a Perl 4 library, timelocal.pl, that was included with Perl 4.036, and was most likely written by Tom Christiansen.

The current version was written by Graham Barr.

BUGS

The whole scheme for interpreting two-digit years can be considered a bug.

Bugs may be submitted at <https://github.com/houseabsolute/Time-Local/issues>.

There is a mailing list available for users of this distribution, <mailto:datetime@perl.org>.

I am also usually active on IRC as 'autarch' on "irc://irc.perl.org".

SOURCE

The source code repository for Time-Local can be found at <https://github.com/houseabsolute/Time-Local>.

AUTHOR

Dave Rolsky <autarch@urth.org>

CONTRIBUTORS

- Florian Ragwitz <rafl@debian.org>
- J. Nick Koston <nick@cpanel.net>
- Unknown <unknown@example.com>

COPYRIGHT AND LICENSE