

Dates and Times

Doug Treder
Beginning Perl

Overview

- Perl grew up in a Unix world, and has a Unixy view of dates and times.
- The concepts ('epoch', 'locale', etc) originally come from the Unix world, but are now pretty universal.
- Everything we'll do here comes with Perl's standard distribution and works the same on all operating systems.

Measuring Time

- Keeping track of time is a pretty popular thing.
- Common time problems are:
 - Comparing two times
 - Figuring out a future or past time
 - Tracking a time interval
- If we could fit a time into a single number, it would be pretty easy to solve all of these.

Enter the Epoch

- Humans tend to think of times like a set of nested boxes or gears in a clock:
 - I'm in the slot called 2007 CE.
 - in the little box on the calendar marked Dec 27
 - it's the second half of the day
 - the little hand is between the 7 and the 8
 - we're in the 35th minute
 - etc

Enter the Epoch

- Not only do people write times differently,
- They even talk about the same time in different ways
- depending on time zone, DST, language, country, and more...

Enter the Epoch

- The bright idea for computers is to keep track of time as a single number.
- Of course the next question is what does ZERO mean?
- On most systems the epoch is 00:00:00 UTC, January 1, 1970
- (A prominent exception being Mac OS Classic which uses 00:00:00, January 1, 1904 in the current local time zone for its epoch.)
- There's no timezone: all epoch times are UTC.

Enter the Epoch

- Epoch times measure the number of seconds from the 'epoch date'
- Negative numbers mark times prior to the epoch (some systems don't allow them though)
- This makes comparing times easy
- Durations and intervals are also just numbers (number of seconds)

Converting Epochs

- So the hardest problems about dates and times are just converting from Human expressions of time into epoch (data coming in) and back (data going out)
- These problems have been solved *a lot* of times.
- TMTOWTDI, but some are better than others
- We'll just talk about the ones that come with Perl.

Current Time

- The current time (according to your computer) can be retrieved with the time built-in keyword:

```
my $start = time;  
  
# processing for a long time....  
  
my $now = time;  
my $duration = $now - $start;  
print "It's been $duration seconds so far...\n";  
  
# this kind of code (measuring durations and intervals)  
# doesn't care what the epoch or locale is
```

localtime

- Perl can convert epoch times to a human expression in the current locale with 'localtime'

```
my ($sec, $min, $hour, $day, $mon, $year, $yday, $yday,  
$dst) = localtime;
```

```
my ($sec, $min, $hour, $day, $mon, $year, $yday, $yday,  
$dst) = localtime $some_epoch_time;
```

```
# if you forget the ordering,  
# consult 'perldoc -f localtime'
```

localtime

- if you don't need part of localtime's output, you can use undef as a left-hand-side placeholder:

```
# all fields:  
my ($sec, $min, $hour, $day, $mon, $year,  
    $yday, $yday, $dst) = localtime;  
  
# just the date:  
my (undef, undef, undef, $day, $mon, $year) = localtime;  
  
# just the time:  
my (undef, $min, $hour) = localtime;
```

Yucky Unix Tricks

- localtime acts like Unix's POSIX standard
- The weekdays start at 0 for Sunday,
- Months are numbered 0 to 11
- Trickiest of all: the year needs 1900 added.
- Perl has no Year-2000 Problem. Why?
- (perl does have a year 2038 problem: why?)

Fixing localtime

- Weekdays and Month Numbers are zero based, to make it easier to convert with arrays:

```
my @weekdays =  
    qw(Sunday Monday Tuesday Wednesday Thursday Friday Saturday);  
my @months =  
    qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);  
  
my (undef, undef, undef, $day, $mon, $year, $wday)  
    = localtime;  
  
$year += 1900;  
print "Today is $weekdays[$wday], $months[$mon] $day, $year\n";
```

Easier localtime

- Time::localtime names the values for you, instead of having to remember the order of the list:

```
use Time::localtime;

my $time = localtime;

print "Today is "
    . $time->day
    . " of the "
    . ($time->mon + 1) . "th month of "
    . $time->year + 1900
    . "\n";
```

Scalar Context

- As a shortcut, localtime in scalar context produces a standard formatted string (but don't forget print is list context!)

```
my $formatted_time = localtime; # scalar context  
my $other_date = localtime( $some_epoch);
```

```
print localtime; # whoops! list context  
6401621110703350
```

```
print scalar localtime; # forced to scalar context  
Sun Dec 2 16:41:34 2007
```

gmtime

- gmtime works the same as localtime, but ignores your locale settings that change the timezone.

```
# the current UTC time
my ($sec, $min, $hour, $day, $mon, $year, $yday, $wday)
= gmtime;

# converting some other UTC time to the list of values:
my ($sec, $min, $hour, $day, $mon, $year, $yday, $wday)
= gmtime( $some_epoch_time );
```

Time::Local

- Going backwards from human values to epoch
- If you have the individual values of the time 'list', and need an epoch,
- Perl comes with the Time::Local module that reverses what localtime and gmtime do

Time::Local

- timelocal reverses what localtime does:

```
# loads the Time::Local module which provides timelocal function
use Time::Local;

# $epoch = timelocal($sec, $min, $hour, $day, $mon, $year)
my $us_santa_arrives
    = timelocal(0,0,0, 25, 11, 2007);

my $uk_saint_nick_arrives
    = timegm(0,0,0, 25, 11, 2007);
```

Solving Common Time Problems

Comparing Times

- The best way to compare two times is using epoch times (or converting to epoch times):

```
use Time::Local;
my $start = time;
# processing for a long time....
my $now = time;
my $duration = $now - $start;
print "It's been $duration seconds so far...\n";

my ($year, $month, $day) = (2007, 12, 1); # 1 Dec 2007
my $then = timelocal(0,0,0, $day, $mon - 1, $year);
$duration = $now - $then;
print "It's been $duration seconds since then!\n";
```

Adding Time

- To get the time for a future or past date, add or subtract epoch times:

```
my $now = time;  
my $SECONDS_IN_A_DAY = 24 * 60 * 60;  
  
my $tomorrow = $now + $SECONDS_IN_A_DAY;  
my $yesterday = $now - $SECONDS_IN_A_DAY;  
  
# if you need to add or subtract times that depend on  
context (i.e. a month, a year (leap days!) or business  
days, try Date::Calc
```

Printing Times

- localtime gives you the individual time list values.
- You can format these for printing any way you like by hand
- But if you only need the individual values for printing a date or time, there are shortcuts.
- The POSIX module provides a Unixy one called strftime:

```
use POSIX qw(strftime);
```

```
print strftime "%a %b %e %H:%M:%S %Y", localtime;
```

```
Sun Dec 2 17:03:29 2007
```

POSIX strftime

- 'man strftime' for the complete list of strftime formats

```
%A    the full weekday name.
%a    the abbreviated weekday name.
%B    the full month name.
%b    the abbreviated month name.
%C    (year / 100) as decimal number;
%c    time and date.
%d    the day of the month as a decimal number(01-31).
%e    the day of month as a decimal number (1-31); single digits with a blank.
%G    year as a decimal number with century.
%g    the same year as in ``%G'', but without century (00-99).
%H    the hour (24-hour clock) as a decimal number (00-23).
%I    the hour (12-hour clock) as a decimal number (01-12).
%j    the day of the year as a decimal number (001-366).
%k    the hour (24-hour clock) as a decimal number (0-23);
%l    the hour (12-hour clock) as a decimal number (1-12);
%M    the minute as a decimal number (00-59).
%m    the month as a decimal number (01-12).
%p    either "ante meridiem" or "post meridiem" as appropriate.
%S    the second as a decimal number (00-60).
%w    the weekday (Sunday as the first day of the week) as a decimal number (0-6).
%Y    the year with century as a decimal number.
%y    the year without century as a decimal number (00-99).
%Z    the time zone name.
%z    the time zone offset from UTC; a leading plus sign
      stands for east of UTC, a minus sign for west of UTC
```

Reading Dates

```
# To read dates in any standard format:
```

```
my $date_string = "2007-12-01";
```

```
# use a regex to pull out the pieces
```

```
my ($year, $month, $day) =  
    $date_string =~ /(\d+)-(\d+)-(\d+)/;
```

```
# then convert to epoch time with timelocal  
use Time::Local;
```

```
my $date = timelocal(0,0,0, $day, $month - 1, $year);
```

```
# don't forget that months are 0 to 11 !
```

Time::HiRes

- The built in time functions in perl work on integers (whole seconds)
- If you need more resolution (up to microsecond - millionths of a second), use Time::HiRes
- Not all functions might work on your machine, but try these examples to test it:

Time::HiRes

```
use Time::HiRes qw(usleep gettimeofday tv_interval);

while (usleep(500000)) {
    print "Twice a second:\n";
}
usleep(50); # sleep 0.000050 seconds

my $start = [gettimeofday]; # arrayref syntax
# processing...
my $seconds = tv_interval($start); # time since $start

# $seconds returned in floating point format:
printf "Processing took %.06f seconds\n", $seconds;
```

More Modules

- `Date::Calc`
- `Date::Parse`
- `Date::Format`
- `DateTime`
- `Date::Manip` (watch out for this one!)