Unix time

From Wikipedia, the free encyclopedia

"Epoch time" redirects here. For the Falun Gong newspaper, see The Epoch Times.
For Wikipedia time function, or #time, see Help:Time function.

Unix time (also known as **Epoch time**, **POSIX time**,[1] **seconds since the Epoch**,[2] or **UNIX Epoch time**[3]) is a system for describing a point in time. It is the number of seconds that have elapsed since the *Unix epoch*, minus leap seconds; the Unix epoch is 00:00:00 UTC on 1 January 1970. Leap seconds are ignored,[4] with a leap second having the same Unix time as the second before it, and every day is treated as if it contains exactly 86 400 seconds.[2] Due to this treatment, Unix time is not a true representation of UTC.

Unix time is widely used in operating systems and file formats. In Unix-like operating systems, `date` is a command which will print or set the current time; by default, it prints or sets the time in the system time zone, but with the `-u` flag, it prints or sets the time in UTC and, with the `TZ` environment variable set to refer to a particular time zone, prints or sets the time in that time zone.[5]

Current Unix time
1596294753 (update)
(2020-08-01T15:12:33+00:00)

Unix time passed 1 000 000 000 seconds on 2001-09-09T01:46:40Z. It was celebrated in Copenhagen, Denmark at a party held by DKUUG (at 03:46:40 local time).

Contents [hide]

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
العربية
Deutsch
Español
Français
한국어
Italiano

Русский
Tiếng Việt
中文
22 more
Edit links

Definition [edit]

Two layers of encoding make up Unix time. The first layer encodes a point in time as a scalar real number which represents the number of seconds that have passed since 00:00:00 UTC Thursday, 1 January 1970.[6] The second layer encodes that number as a sequence of bits or decimal digits.

As is standard with UTC, this article labels days using the Gregorian calendar, and counts times within each day in hours, minutes, and seconds. Some of the examples also show International Atomic Time (TAI), another time scheme which uses the same seconds and is displayed in the same format as UTC, but in which every day is exactly 86 400 seconds long, gradually losing synchronization with the Earth's rotation at a rate of roughly one second per year.

Encoding time as a number [edit]

Unix time is a single signed number that increments every second, which makes it easier for computers to store and manipulate than conventional date systems. Interpreter programs can then convert it to a human-readable format.

The *Unix epoch* is the time 00:00:00 UTC on 1 January 1970.[2] There is a problem with this definition, in that UTC did not exist in its current form until 1972; this issue is discussed below. For brevity, the remainder of this section uses ISO 8601 date and time format, in which the Unix epoch is 1970-01-01T00:00:00Z.

The Unix time number is zero at the Unix epoch, and increases by exactly 86 400 per day since the epoch. Thus 2004-09-16T00:00:00Z, 12 677 days after the epoch, is represented by the Unix time number 12 677 × 86 400 = 1 095 292 800. This can be extended backwards from the epoch too, using negative numbers; thus 1957-10-04T00:00:00Z, 4472 days before the epoch, is represented by the Unix time number −4472 × 86 400 = −386 380 800. This applies within days as well; the time number at any given time of a day is the number of seconds that has passed since the midnight starting that day added to the time number of that midnight.

Because Unix time is based on an epoch, and because of a common misunderstanding that the Unix epoch is the only epoch (often called "*the Epoch*" [2]), Unix time is sometimes referred to as *Epoch time*.[7][8][9]

Leap seconds [edit]

The above scheme means that on a normal UTC day, which has a duration of 86 400 seconds, the Unix time number changes in a continuous manner across midnight. For example, at the end of the day used in the examples above, the time representations progress as follows:

Unix time across midnight into 17 September 2004 (no leap second)

TAI (17 September 2004)	UTC (16 to 17 September 2004)	Unix time
2004-09-17T00:00:30.75	2004-09-16T23:59:58.75	1 095 379 198.75
2004-09-17T00:00:31.00	2004-09-16T23:59:59.00	1 095 379 199.00
2004-09-17T00:00:31.25	2004-09-16T23:59:59.25	1 095 379 199.25
2004-09-17T00:00:31.50	2004-09-16T23:59:59.50	1 095 379 199.50
2004-09-17T00:00:31.75	**2004-09-16T23:59:59.75**	**1 095 379 199.75**
2004-09-17T00:00:32.00	**2004-09-17T00:00:00.00**	**1 095 379 200.00**
2004-09-17T00:00:32.25	2004-09-17T00:00:00.25	1 095 379 200.25

2004-09-17T00:00:32.50	2004-09-17T00:00:00.50	1 095 379 200.50
2004-09-17T00:00:32.75	2004-09-17T00:00:00.75	1 095 379 200.75
2004-09-17T00:00:33.00	2004-09-17T00:00:01.00	1 095 379 201.00
2004-09-17T00:00:33.25	2004-09-17T00:00:01.25	1 095 379 201.25

When a leap second occurs, the UTC day is not exactly 86 400 seconds long and the Unix time number (which always increases by exactly 86 400 each day) experiences a discontinuity. Leap seconds may be positive or negative. No negative leap second has ever been declared, but if one were to be, then at the end of a day with a negative leap second, the Unix time number would jump up by 1 to the start of the next day. During a positive leap second at the end of a day, which occurs about every year and a half on average, the Unix time number increases continuously into the next day during the leap second and then at the end of the leap second jumps back by 1 (returning to the start of the next day). For example, this is what happened on strictly conforming POSIX.1 systems at the end of 1998:

Unix time across midnight into 1 January 1999 (positive leap second)

TAI (1 January 1999)	UTC (31 December 1998 to 1 January 1999)	Unix time
1999-01-01T00:00:29.75	1998-12-31T23:59:58.75	915 148 798.75
1999-01-01T00:00:30.00	1998-12-31T23:59:59.00	915 148 799.00
1999-01-01T00:00:30.25	1998-12-31T23:59:59.25	915 148 799.25
1999-01-01T00:00:30.50	1998-12-31T23:59:59.50	915 148 799.50
1999-01-01T00:00:30.75	**1998-12-31T23:59:59.75**	**915 148 799.75**
1999-01-01T00:00:31.00	**1998-12-31T23:59:60.00**	**915 148 800.00**
1999-01-01T00:00:31.25	1998-12-31T23:59:60.25	915 148 800.25
1999-01-01T00:00:31.50	1998-12-31T23:59:60.50	915 148 800.50
1999-01-01T00:00:31.75	**1998-12-31T23:59:60.75**	**915 148 800.75**
1999-01-01T00:00:32.00	**1999-01-01T00:00:00.00**	**915 148 800.00**
1999-01-01T00:00:32.25	1999-01-01T00:00:00.25	915 148 800.25
1999-01-01T00:00:32.50	1999-01-01T00:00:00.50	915 148 800.50
1999-01-01T00:00:32.75	1999-01-01T00:00:00.75	915 148 800.75
1999-01-01T00:00:33.00	1999-01-01T00:00:01.00	915 148 801.00
1999-01-01T00:00:33.25	1999-01-01T00:00:01.25	915 148 801.25

Unix time numbers are repeated in the second immediately following a positive leap second. The Unix time number 1 483 142 400 is thus ambiguous: it can refer either to start of the leap second (2016-12-31 23:59:60) or the end of it, one second later (2017-01-01 00:00:00). In the theoretical case when a negative leap second occurs, no ambiguity is caused, but instead there is a range of Unix time numbers that do not refer to any point in UTC time at all.

A Unix clock is often implemented with a different type of positive leap second handling associated with the Network Time Protocol (NTP). This yields a system that does not conform to the POSIX standard. See the section below concerning NTP for details.

When dealing with periods that do not encompass a UTC leap second, the difference between two Unix time numbers is equal to the duration in seconds of the period between the corresponding points in time. This is a common computational technique. However, where leap seconds occur, such calculations give the wrong answer. In applications where this level of accuracy is required, it is necessary to consult a table of leap seconds when dealing with Unix times, and it is often preferable to use a different time encoding that does not suffer from this problem.

A Unix time number is easily converted back into a UTC time by taking the quotient and modulus of the Unix time number, modulo 86 400. The quotient is the number of days since the epoch, and the modulus is the number of seconds since midnight UTC on that day. If given a Unix time number that is ambiguous due to a positive leap second, this algorithm interprets it as the time just after midnight. It never generates a time that is during a leap second. If given a Unix time number that is invalid due to a negative leap second, it generates an equally invalid UTC time. If these conditions are significant, it is necessary to consult a table of leap seconds to detect them.

Non-synchronous Network Time Protocol-based variant [edit]

Commonly a Mills-style Unix clock is implemented with leap second handling not synchronous with the change of the Unix time number. The time number initially decreases where a leap should have occurred, and then it leaps to the correct time 1 second after the leap. This makes implementation easier, and is described by Mills' paper.[10] This is what happens across a positive leap second:

Non-synchronous Mills-style Unix clock across midnight into 1 January 1999 (positive leap second)

TAI (1 January 1999)	UTC (31 December 1998 to 1 January 1999)	State	Unix clock
1999-01-01T00:00:29.75	1998-12-31T23:59:58.75	TIME_INS	915 148 798.75
1999-01-01T00:00:30.00	1998-12-31T23:59:59.00	TIME_INS	915 148 799.00
1999-01-01T00:00:30.25	1998-12-31T23:59:59.25	TIME_INS	915 148 799.25
1999-01-01T00:00:30.50	1998-12-31T23:59:59.50	TIME_INS	915 148 799.50
1999-01-01T00:00:30.75	1998-12-31T23:59:59.75	TIME_INS	915 148 799.75
1999-01-01T00:00:31.00	**1998-12-31T23:59:60.00**	**TIME_INS**	**915 148 800.00**
1999-01-01T00:00:31.25	**1998-12-31T23:59:60.25**	**TIME_OOP**	**915 148 799.25**
1999-01-01T00:00:31.50	1998-12-31T23:59:60.50	TIME_OOP	915 148 799.50
1999-01-01T00:00:31.75	**1998-12-31T23:59:60.75**	**TIME_OOP**	**915 148 799.75**
1999-01-01T00:00:32.00	**1999-01-01T00:00:00.00**	**TIME_OOP**	**915 148 800.00**
1999-01-01T00:00:32.25	1999-01-01T00:00:00.25	TIME_WAIT	915 148 800.25
1999-01-01T00:00:32.50	1999-01-01T00:00:00.50	TIME_WAIT	915 148 800.50
1999-01-01T00:00:32.75	1999-01-01T00:00:00.75	TIME_WAIT	915 148 800.75
1999-01-01T00:00:33.00	1999-01-01T00:00:01.00	TIME_WAIT	915 148 801.00

1999-01-01T00:00:33.25	1999-01-01T00:00:01.25	TIME_WAIT	915 148 801.25

This can be decoded properly by paying attention to the leap second state variable, which unambiguously indicates whether the leap has been performed yet. The state variable change is synchronous with the leap.

A similar situation arises with a negative leap second, where the second that is skipped is slightly too late. Very briefly the system shows a nominally impossible time number, but this can be detected by the **TIME_DEL** state and corrected.

In this type of system the Unix time number violates POSIX around both types of leap second. Collecting the leap second state variable along with the time number allows for unambiguous decoding, so the correct POSIX time number can be generated if desired, or the full UTC time can be stored in a more suitable format.

The decoding logic required to cope with this style of Unix clock would also correctly decode a hypothetical POSIX-conforming clock using the same interface. This would be achieved by indicating the **TIME_INS** state during the entirety of an inserted leap second, then indicating **TIME_WAIT** during the entirety of the following second while repeating the seconds count. This requires synchronous leap second handling. This is probably the best way to express UTC time in Unix clock form, via a Unix interface, when the underlying clock is fundamentally untroubled by leap seconds.

TAI-based variant [edit]

This section's **factual accuracy is disputed**. Relevant discussion may be found on Talk:Unix time. Please help to ensure that disputed statements are reliably sourced. *(April 2016) (Learn how and when to remove this template message)*

Another, much rarer, non-conforming variant of Unix time keeping involves encoding TAI rather than UTC; some Linux systems are configured this way.[11] Because TAI has no leap seconds, and every TAI day is exactly 86400 seconds long, this encoding is actually a pure linear count of seconds elapsed since 1970-01-01T00:00:00 TAI. This makes time interval arithmetic much easier. Time values from these systems do not suffer the ambiguity that strictly conforming POSIX systems or NTP-driven systems have.

In these systems it is necessary to consult a table of leap seconds to correctly convert between UTC and the pseudo-Unix-time representation. This resembles the manner in which time zone tables must be consulted to convert to and from civil time; the IANA time zone database includes leap second information, and the sample code available from the same source uses that information to convert between TAI-based time stamps and local time. Conversion also runs into definitional problems prior to the 1972 commencement of the current form of UTC (see section UTC basis below).

This TAI-based system, despite its superficial resemblance, is not Unix time. It encodes times with values that differ by several seconds from the POSIX time values.

Representing the number [edit]

A Unix time number can be represented in any form capable of representing numbers. In some applications the number is simply represented textually as a string of decimal digits, raising only trivial additional problems. However, certain binary representations of Unix times are particularly significant.

The Unix `time_t` data type that represents a point in time is, on many platforms, a signed integer, traditionally of 32 bits (but see below), directly encoding the Unix time number as described in the preceding section. Being 32 bits means that it covers a range of about 136 years in total. The minimum

representable date is Friday 1901-12-13, and the maximum representable date is Tuesday 2038-01-19. One second after 03:14:07 UTC 2038-01-19 this representation will overflow. This milestone is anticipated with a mixture of amusement and dread—see year 2038 problem.

In some newer operating systems, `time_t` has been widened to 64 bits. This expands the times representable by approximately 293 billion years in both directions, which is over twenty times the present age of the universe per direction.

There was originally some controversy over whether the Unix `time_t` should be signed or unsigned. If unsigned, its range in the future would be doubled, postponing the 32-bit overflow (by 68 years). However, it would then be incapable of representing times prior to the epoch. The consensus is for `time_t` to be signed, and this is the usual practice. The software development platform for version 6 of the QNX operating system has an unsigned 32-bit `time_t`, though older releases used a signed type.

The POSIX and Open Group Unix specifications include the C standard library, which includes the time types and functions defined in the `<time.h>` header file. The ISO C standard states that `time_t` must be an arithmetic type, but does not mandate any specific type or encoding for it. POSIX requires `time_t` to be an integer type, but does not mandate that it be signed or unsigned.

Unix has no tradition of directly representing non-integer Unix time numbers as binary fractions. Instead, times with sub-second precision are represented using composite data types that consist of two integers, the first being a `time_t` (the integral part of the Unix time), and the second being the fractional part of the time number in millionths (in `struct timeval`) or billionths (in `struct timespec`).[12][13] These structures provide a decimal-based fixed-point data format, which is useful for some applications, and trivial to convert for others.

UTC basis [edit]

The present form of UTC, with leap seconds, is defined only starting from 1 January 1972. Prior to that, since 1 January 1961 there was an older form of UTC in which not only were there occasional time steps, which were by non-integer numbers of seconds, but also the UTC second was slightly longer than the SI second, and periodically changed to continuously approximate the Earth's rotation. Prior to 1961 there was no UTC, and prior to 1958 there was no widespread atomic timekeeping; in these eras, some approximation of GMT (based directly on the Earth's rotation) was used instead of an atomic timescale.[*citation needed*]

The precise definition of Unix time as an encoding of UTC is only uncontroversial when applied to the present form of UTC. The Unix epoch predating the start of this form of UTC does not affect its use in this era: the number of days from 1 January 1970 (the Unix epoch) to 1 January 1972 (the start of UTC) is not in question, and the number of days is all that is significant to Unix time.

The meaning of Unix time values below +63 072 000 (i.e., prior to 1 January 1972) is not precisely defined. The basis of such Unix times is best understood to be an unspecified approximation of UTC. Computers of that era rarely had clocks set sufficiently accurately to provide meaningful sub-second timestamps in any case. Unix time is not a suitable way to represent times prior to 1972 in applications requiring sub-second precision; such applications must, at least, define which form of UT or GMT they use.

As of 2009, the possibility of ending the use of leap seconds in civil time is being considered.[14] A likely means to execute this change is to define a new time scale, called *International Time*, that initially matches UTC but thereafter has no leap seconds, thus remaining at a constant offset from TAI. If this happens, it is likely that Unix time will be prospectively defined in terms of this new time scale, instead of UTC. Uncertainty about whether this will occur makes prospective Unix time no less predictable than it already is: if UTC were simply to have no further leap seconds the result would be the same.

History [edit]

This section **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(September 2019) (Learn how and when to remove this template message)*

The earliest versions of Unix time had a 32-bit integer incrementing at a rate of 60 Hz, which was the rate of the system clock on the hardware of the early Unix systems. The value 60 Hz still appears in some software interfaces as a result. The epoch also differed from the current value. The first edition Unix Programmer's Manual dated 3 November 1971 defines the Unix time as "the time since 00:00:00, 1 January 1971, measured in sixtieths of a second".[15]

The User Manual also commented that "the chronologically-minded user will note that 2**32 sixtieths of a second is only about 2.5 years". Because of this limited range, the epoch was redefined more than once, before the rate was changed to 1 Hz and the epoch was set to its present value of 1 January 1970 00:00:00 UTC. This yielded a range of about 136 years, half of it before 1970 and half of it afterwards.

As indicated by the definition quoted above, the Unix time scale was originally intended to be a simple linear representation of time elapsed since an epoch. However, there was no consideration of the details of time scales, and it was implicitly assumed that there was a simple linear time scale already available and agreed upon. The first edition manual's definition does not even specify which time zone is used. Several later problems, including the complexity of the present definition, result from Unix time having been defined gradually by usage rather than fully defined from the outset.

When POSIX.1 was written, the question arose of how to precisely define `time_t` in the face of leap seconds. The POSIX committee considered whether Unix time should remain, as intended, a linear count of seconds since the epoch, at the expense of complexity in conversions with civil time or a representation of civil time, at the expense of inconsistency around leap seconds. Computer clocks of the era were not sufficiently precisely set to form a precedent one way or the other.

The POSIX committee was swayed by arguments against complexity in the library functions,[*citation needed*] and firmly defined the Unix time in a simple manner in terms of the elements of UTC time. This definition was so simple that it did not even encompass the entire leap year rule of the Gregorian calendar, and would make 2100 a leap year.

The 2001 edition of POSIX.1 rectified the faulty leap year rule in the definition of Unix time, but retained the essential definition of Unix time as an encoding of UTC rather than a linear time scale. Since the mid-1990s, computer clocks have been routinely set with sufficient precision for this to matter, and they have most commonly been set using the UTC-based definition of Unix time. This has resulted in considerable complexity in Unix implementations, and in the Network Time Protocol, to execute steps in the Unix time number whenever leap seconds occur.[*citation needed*]

Notable events in Unix time [edit]

Unix enthusiasts have a history of holding "time_t parties" (pronounced "time tea parties") to celebrate significant values of the Unix time number.[16][17] These are directly analogous to the new year celebrations that occur at the change of year in many calendars. As the use of Unix time has spread, so has the practice of celebrating its milestones. Usually it is time values that are round numbers in decimal that are celebrated, following the Unix convention of viewing `time_t` values in decimal. Among some groups round binary numbers are also celebrated, such as $+2^{30}$ which occurred at 13:37:04 UTC on Saturday, 10 January 2004.[*citation needed*]

The events that these celebrate are typically described as "*N* seconds since the Unix epoch", but this is inaccurate; as discussed above, due to the handling of leap seconds in Unix time the number of seconds elapsed since the Unix epoch is slightly greater than the Unix time number for times later than the epoch.

- At 18:36:57 UTC on Wednesday, 17 October 1973, the first appearance of the date in ISO 8601 format[a] (1973-10-17) within the digits of Unix time (119731017) took place.
- At 01:46:40 UTC on Sunday, 9 September 2001, the Unix billennium (Unix time number 1 000 000 000) was celebrated.[18] The name *billennium* is a portmanteau of *billion* and *millennium*.[19][20] Some programs which stored timestamps using a text representation encountered sorting errors, as in a text sort times after the turnover, starting with a *1* digit, erroneously sorted before earlier times starting with a *9* digit. Affected programs included the popular Usenet reader KNode and e-mail client KMail, part of the KDE desktop environment. Such bugs were generally cosmetic in nature and quickly fixed once problems became apparent. The problem also affected many *Filtrix* document-format filters provided with Linux versions of WordPerfect; a patch was created by the user community to solve this problem, since Corel no longer sold or supported that version of the program.[21]
- At 23:31:30 UTC on Friday, 13 February 2009, the decimal representation of Unix time reached 1 234 567 890 seconds.[22] Google celebrated this with a Google doodle.[23] Parties and other celebrations were held around the world, among various technical subcultures, to celebrate the 1 234 567 890th second.[16][24]
- At 03:33:20 UTC on Wednesday, 18 May 2033, the Unix time value will equal 2 000 000 000 seconds.
- At 06:28:16 UTC on Thursday, 7 February 2036, Network Time Protocol will loop over to the next epoch, as the 32-bit time stamp value used in NTP (unsigned, but based on 1 January 1900) will overflow. This date is close to the following date because the 136-year range of a 32-bit integer number of seconds is close to twice the 70-year offset between the two epochs.
- At 03:14:08 UTC on Tuesday, 19 January 2038, 32-bit versions of the Unix time stamp will cease to work, as it will overflow the largest value that can be held in a signed 32-bit number ($7FFFFFFF_{16}$ or 2 147 483 647). Before this moment, software using 32-bit time stamps will need to adopt a new convention for time stamps,[25] and file formats using 32-bit time stamps will need to be changed to support larger time stamps or a different epoch. If unchanged, the next second will be incorrectly interpreted as 20:45:52 Friday 13 December 1901 UTC. This is referred to as the Year 2038 problem.
- At 05:20:00 UTC on Saturday, 24 January 2065, the Unix time value will equal 3 000 000 000 seconds.
- At 06:28:15 UTC on Sunday, 7 February 2106, the Unix time will reach $FFFFFFFF_{16}$ or 4 294 967 295 seconds which, for systems that hold the time on 32-bit unsigned integers, is the maximum attainable. For some of these systems, the next second will be incorrectly interpreted as 00:00:00 Thursday 1 January 1970 UTC. Other systems may experience an overflow error with unpredictable outcomes.[*citation needed*]
- At 15:30:08 UTC on Sunday, 4 December 292 277 026 596,[26][27] 64-bit versions of the Unix time stamp cease to work, as it will overflow the largest value that can be held in a signed 64-bit number. This is nearly 22 times the estimated current age of the universe, which is 1.37×10^{10} years (13.7 billion).

In literature and calendrics [edit]

Vernor Vinge's novel *A Deepness in the Sky* describes a spacefaring trading civilization thousands of years in the future that still uses the Unix epoch. The "programmer-archaeologist" responsible for finding and maintaining usable code in mature computer systems first believes that the epoch refers to the time when man first walked on the Moon, but then realizes that it is "the 0-second of one of Humankind's first computer operating systems".[28]

See also [edit]

- Epoch (computing)
- System time

Notes [edit]

a. ^ cited retroactively since ISO 8601 was published in 1988.

References [edit]

1. ^ "The Open Group Base Specifications Issue 7, Rationale: Base Definitions, section A.4 General Concepts". The Open Group. Retrieved 9 September 2019.
2. ^ a b c d "The Open Group Base Specifications Issue 7, section 4.16 Seconds Since the Epoch". The Open Group. Retrieved 22 January 2017.
3. ^ Matthew, Neil; Stones, Richard (2008). "The Linux Environment". *Beginning Linux Programming*. Indianapolis, Indiana, US: Wiley. p. 148. ISBN 978-0-470-14762-7.
4. ^ "The Open Group Base Specifications Issue 7, Rationale, section 4.16 Seconds Since the Epoch". The OpenGroup. Retrieved 22 January 2017.
5. ^ `date` – Commands & Utilities Reference, The Single UNIX Specification, Issue 7 from The Open Group
6. ^ "Epoch Converter - Unix Timestamp Converter". *Epoch Converter*. Retrieved 12 January 2020.
7. ^ "Handling timestamps using Format Date in Shortcuts". *Apple Inc*. Retrieved 19 June 2019.
8. ^ "RAW date format in CSV exported reports". *International Business Machines Corporation (IBM)*. Retrieved 19 June 2019.
9. ^ "TIMESTAMP BY (Azure Stream Analytics)". *Microsoft Corporation*. Retrieved 19 June 2019.
10. ^ Mills, David L. (12 May 2012). "The NTP Timescale and Leap Seconds". *eecis.udel.edu*. Retrieved 21 August 2017.
11. ^ "Time Scales". *Network Time Protocol Wiki*. 24 July 2019. Retrieved 12 January 2020.
12. ^ "timespec". *NetBSD Manual Pages*. 12 April 2011. Retrieved 5 July 2019.
13. ^ "time.h(0P)". *Linux manual page*. Retrieved 5 July 2019.
14. ^ McCarthy, D. D.; Seidelmann, P. K. (2009). *TIME—From Earth Rotation to Atomic Physics*. Weinheim: Wiley–VCH Verlag GmbH & Co. KGaA. p. 232. ISBN 978-3-527-40780-4.
15. ^ *Unix Programmer's Manual* (PDF) (1st ed.). 3 November 1971. Retrieved 28 March 2012. "*time* returns the time since 00:00:00, 1 Jan. 1971, measured in sixtieths of a second."
16. ^ a b Tweney, Dylan (12 February 2009). "Unix Lovers to Party Like It's 1234567890". *Wired*.
17. ^ "Slashdot | date +%s Turning 1111111111". 17 March 2005.[*unreliable source?*]
18. ^ "Unix time facts & trivia – Unix Time . Info". Archived from the original on 27 October 2017.
19. ^ "UNIX Approaches Ripe Old Age of One Billion". Electromagnetic.net. Archived from the original on 13 April 2013. Retrieved 6 December 2012.
20. ^ "The Risks Digest Volume 21: Issue 69". Catless.ncl.ac.uk. Retrieved 6 December 2012.
21. ^ "Technical Problems". *linuxmafia.com*. Retrieved 21 August 2017.
22. ^ nixCraft. "Humor: On Feb, Friday 13, 2009 Unix time Will Be 1234567890". Cyberciti.biz. Retrieved 6 December 2012.
23. ^ "Google 1234567890 Logo". Google Inc. Retrieved 28 January 2013.
24. ^ Ahmed, Murad (13 February 2009). "At the third stroke, the Unix time will be 1234567890". *The Times*.
25. ^ "Unix Time Stamp.com". UnixTimeStamp.com. Retrieved 12 January 2020.
26. ^ Spinellis, Diomidis (7 April 2006). "Code Quality: The Open Source Perspective". ISBN 9780321166074.
27. ^ IDRBT Working Paper No. 9 Y2K38 – Ashutosh Saxena and Sanjay Rawat
28. ^ Mashey, John R. (27 December 2004). "Languages, Levels, Libraries, and Longevity". *Queue*. **2** (9): 32–38.

External links [edit]

- Unix Programmer's Manual, first edition
- Personal account of the POSIX decisions by Landon Curt Noll
- Clewett, James. *2,147,483,647 – The End of Time [Unix]*.
- Unix Timestamp Converter

- chrono-Compatible Low-Level Date Algorithms – algorithms to convert between Gregorian and Julian dates and the number of days since the start of Unix time

v·t·e	Calendars	[hide]
Systems	Lunar · Lunisolar · Solar	
In wide use	Astronomical · Chinese · Gregorian · Hindu · Islamic · ISO · **Unix time**	
In more limited use	Akan · Armenian · Assamese (Bhāshkarābda) · Assyrian · Bahá'í (Badí') · Balinese pawukon · Balinese saka · Bengali · Bangladeshi · Berber · Buddhist · Burmese · Chinese (Earthly Branches · Heavenly Stems) · Ethiopian · Gaelic · Germanic heathen · Georgian · Hebrew · Hindu or Indian (Vikram Samvat · Saka) · Igbo · Iranian (Jalali (medieval) · Hijri (modern) · Zoroastrian) · Islamic (Fasli · Tabular) · Jain · Japanese · Javanese · Korean (*Juche*) · Kurdish · Lithuanian · Malayalam · Manipuri (Meitei) · Melanau · Mongolian · Nanakshahi · Nepal Sambat · Nisga'a · Oromo · Romanian · Somali · Sesotho · Slavic (Slavic Native Faith) · Tamil · Taiwanese · Thai (lunar · solar) · Tibetan · Vietnamese · Xhosa · Yoruba	
	Types	Runic · Mesoamerican (Long Count · Calendar round)
	Christian variants	Coptic · Julian (Revised) · Liturgical year (Eastern Orthodox) · Saints
Historical	Arabian · Attic · Aztec (Tōnalpōhualli · Xiuhpōhualli) · Babylonian · Bulgar · Byzantine · Cappadocian · Celtic · Cham · Culāsakaraj · Egyptian · Florentine · French Republican · Germanic · Greek · Hindu · Inca · Macedonian · Maya (Haab' · Tzolk'in) · Muisca · Pentecontad · Pisan · Rapa Nui · Roman calendar · Rumi · Soviet · Swedish · Turkmen	
By specialty	Holocene (anthropological) · Proleptic Gregorian / Proleptic Julian (historiographical) · Darian (Martian) · Dreamspell (New Age) · Discordian · 'Pataphysical	
Proposals	Hanke–Henry Permanent · International Fixed · Pax · Positivist · Symmetry454 · World	
Fictional	Discworld (*Discworld*) · Greyhawk (*Dungeons & Dragons*) · Middle-earth (*The Lord of the Rings*) · Stardate (*Star Trek*) · Galactic Standard Calendar (*Star Wars*)	
Displays and applications	Electronic · Perpetual · Wall	
Year naming and numbering	Terminology	Era · Epoch · Regnal name · Regnal year · Year zero
	Systems	Ab urbe condita · Anno Domini/Common Era · Anno Mundi · Assyrian · Before Present · Chinese Imperial · Chinese Minguo · Human Era · Japanese · Korean · Seleucid · Spanish · Yugas (Satya · Treta · Dvapara · Kali) · Vietnamese
	List of calendars · Category	

Categories: Calendaring standards | Network time-related software | Time measurement systems | Time scales | Unix | 1970