System time - ArchWiki

Archived: 2026-04-05 17:54:23 UTC

 **This article or section needs expansion.**

Reason: This article mostly documents systemd *timedatectl*; explain basic commands like *date* and *hwclock* first (Discuss in Talk:System time)

In an operating system, the time (clock) is determined by three parts: time value, whether it is local time or UTC or something else, time zone, and Daylight Saving Time *(DST)* if applicable. This article explains what they are and how to read/set them. Two clocks are present on systems: a hardware clock and a system clock which are also detailed in this article.

Standard behavior of most operating systems is:

- Set the system clock from the hardware clock on boot.
- Keep accurate time of the system clock, see #Time synchronization.
- Set the hardware clock from the system clock on shutdown.

Time standard

There are two time standards: **localtime** and Coordinated Universal Time (**UTC**). The localtime standard is dependent on the current *time zone*, while UTC is the *global* time standard and is independent of time zone values. Though conceptually different, UTC is also known as GMT (Greenwich Mean Time).

The standard used by the hardware clock (CMOS clock, the BIOS time) is set by the operating system. By default, Windows uses localtime, macOS uses UTC, other UNIX and UNIX-like systems vary. An OS that uses the UTC standard will generally consider the hardware clock as UTC and make an adjustment to it to set the OS time at boot according to the time zone.

Hardware clock

The **hardware clock** (a.k.a. the Real Time Clock (RTC) or CMOS clock) stores the values of: Year, Month, Day, Hour, Minute, and Seconds. A UEFI firmware has the additional ability to store the timezone, and whether DST is used.

Read hardware clock

```
# hwclock --show
```

Set hardware clock from system clock

The following sets the hardware clock from the system clock. Additionally it updates `/etc/adjtime` or creates it if not present. See hwclock(8) § The Adjtime File for more information on this file as well as the #Time skew section.

```
# hwclock --systohc
```

Automatic syncing

By default, Arch Linux kernels have a feature enabled where the hardware clock is synchronized to the system clock every 11 minutes. You can see if this is enabled on your kernel as follows:

```
$ zgrep CMOS /proc/config.gz
```

```
CONFIG_GENERIC_CMOS_UPDATE=y
CONFIG_RTC_DRV_CMOS=y
```

The first synchronization happens at boot time. What this means is that if your hardware clock is extremely out of date (for example, a CMOS battery failure has reset the clock to the year 2000) then for the first 11 minutes after boot anything which requires a reasonably accurate time will give an error - including SSL, uses the Online Certificate Status Protocol (OCSP). A web browser running on your computer typically sends the hardware clock time in its requests to websites, and a time which is too far out will result in the browser refusing to connect because of an OCSP error.

System clock

The **system clock** (a.k.a. the software clock) keeps track of: time, time zone, and DST if applicable. It is calculated by the Linux kernel as the number of seconds since midnight January 1st 1970, UTC. The initial value of the system clock is calculated from the hardware clock, dependent on the contents of `/etc/adjtime`. After boot-up has completed, the system clock runs independently of the hardware clock. The Linux kernel keeps track of the system clock by counting timer interrupts.

Read clock

To check the current system clock time (presented both in local time and UTC) as well as the RTC (hardware clock):

```
$ timedatectl
```

Set system clock

To set the local time of the system clock directly:

```
# timedatectl set-time "yyyy-MM-dd hh:mm:ss"
```

For example:

```
# timedatectl set-time "2014-05-26 11:13:54"
```

sets the time to May 26th, year 2014, 11:13 and 54 seconds.

Multiple systems

If multiple operating systems are installed on a machine, they will all derive the current time from the same hardware clock: it is recommended to set it to UTC to avoid conflicts across systems. Otherwise, if the hardware clock is set to *localtime*, more than one operating system may adjust it after a DST change for example, thus resulting in an over-correction; problems may also arise when traveling between different time zones and using one of the operating systems to reset the system/hardware clock.

The hardware clock can be queried and set with the `timedatectl` command. You can see the current hardware clock time standard of the Arch system using:

```
$ timedatectl | grep local
```

```
RTC in local TZ: no
```

To change the hardware clock time standard to localtime, use:

```
# timedatectl set-local-rtc 1
```

To revert to the hardware clock being in UTC, type:

```
# timedatectl set-local-rtc 0
```

These generate `/etc/adjtime` automatically and update the RTC accordingly; no further configuration is required.

During kernel startup, at the point when the RTC driver is loaded, the system clock may be set from the hardware clock. Whether this occurs depends on the hardware platform, the version of the kernel and kernel build options. If this does occur, at this point in the boot sequence, the hardware clock time is assumed to be UTC and the value of `/sys/class/rtc/rtc`*N*`/hctosys` (N=0,1,2,..) will be set to 1.

Later, the system clock is set again from the hardware clock by systemd, dependent on values in `/etc/adjtime`. Hence, having the hardware clock using localtime may cause some unexpected behavior during the boot sequence; e.g system time going backwards, which is always a bad idea (there is a lot more to it). Since systemd version 216,

when the RTC is configured to the local time (rather than UTC) systemd will never synchronize back to it, as this might confuse Windows at a later boot. And systemd will no longer inform the kernel about the current timezone. This hence means FAT timestamps will be always considered UTC[1].

Note

- The use of `timedatectl` requires an active D-Bus. Therefore, it may not be possible to use this command under a chroot (such as during installation). In these cases, you can revert back to the hwclock command, or use systemd-nspawn instead of chroot.
- If `/etc/adjtime` is not present, systemd assumes the hardware clock is set to UTC.

UTC in Microsoft Windows

To dual boot with Windows, it is recommended to configure Windows to use UTC, rather than Linux to use localtime. (Windows by default uses localtime [2].)

It can be done by a simple registry fix: Open `regedit` and add a `DWORD` value with hexadecimal value `1` to the registry `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation\RealTimeIsUniversal`

You can do this from an Administrator Command Prompt running:

```
C:\>reg add "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\TimeZoneInformation" /v RealTimeIsU
```

Alternatively, create a `*.reg` file (on the desktop) with the following content and double-click it to import it into registry:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation]
"RealTimeIsUniversal"=dword:00000001
```

Should Windows ask to update the clock due to DST changes, let it. It will leave the clock in UTC as expected, only correcting the displayed time.

The #Hardware clock and #System clock time may need to be updated after setting this value.

If you are having issues with the offset of the time, try reinstalling tzdata and then setting your time zone again:

```
# timedatectl set-timezone America/Los_Angeles
```

UTC in Ubuntu/Fedora

Many Linux distributions have the hardware clock set to be interpreted as in "localtime" if Windows was detected on any disk during their installation. This is apparently done deliberately to allow new users to try out Linux on their Windows computers without editing the registry.

For changing this behavior, see above.

Multi-NTP interaction

If you use an NTP client (see #Time synchronization below) that keeps track of RTC drift on any system, you should disable time synchronization on all but one system. Otherwise the NTP clients would be unaware of each other's adjustment and make grossly incorrect estimates of the RTC drift.

For Windows, go to the *Date and time settings* and uncheck the time sync option. You can also run `w32tm /unregister` as an administrator to unregister the time-sync service: Active Directory machines are known to ignore the synchronization settings and perform a synchronization anyways to prevent replay attacks. The Windows clock synchronization routine is quite inaccurate to start with, requiring even extra work to reach one-second accuracy, so disabling it should not be much of a loss.

Time zone

To check the current zone defined for the system:

```
$ timedatectl status
```

To list available zones:

```
$ timedatectl list-timezones
```

To set your time zone:

```
# timedatectl set-timezone Area/Location
```

Where *`Area`* is a continent or ocean, and *`Location`* is a specific location within the area. North and South America share the same area— `America` .[3]

Example:

```
# timedatectl set-timezone America/Toronto
```

This will create an `/etc/localtime` symlink that points to a zoneinfo file under `/usr/share/zoneinfo/` . In case you choose to create the link manually (for example during chroot where `timedatectl` will not work), keep in mind that it must be a symbolic link, as specified in localtime(5) § DESCRIPTION:

```
# ln -sf /usr/share/zoneinfo/Area/Location /etc/localtime
```

Tip The time zone can also be selected interactively with *tzselect*.

See timedatectl(1) and localtime(5) for details.

Setting based on geolocation

Note Some desktop environments have support for automatic time zone selection (e.g. see GNOME#Date & time).

To set the timezone automatically based on the IP address location, one can use a geolocation API to retrieve the timezone, for example `curl https://ipapi.co/timezone`, and pass the output to `timedatectl set-timezone` for automatic setting. Some geo-IP APIs that provide free or partly free services are listed below:

- Abstract IP geolocation API
- FreegeoIP
- IP-api
- IPAPI
- Ipdata
- Ipstack

Update timezone every time NetworkManager connects to a network

See NetworkManager#Automatically set the timezone.

Time skew

Every clock has a value that differs from *real time* (the best representation of which being International Atomic Time); no clock is perfect. A quartz-based electronic clock keeps imperfect time, but maintains a consistent inaccuracy. This base 'inaccuracy' is known as 'time skew' or 'time drift'.

When the hardware clock is set with `hwclock`, a new drift value is calculated in seconds per day. The drift value is calculated by using the difference between the new value set and the hardware clock value just before the set, taking into account the value of the previous drift value and the last time the hardware clock was set. The new drift value and the time when the clock was set is written to the file `/etc/adjtime` overwriting the previous values. The hardware clock can therefore be adjusted for drift when the command `hwclock --adjust` is run; this also occurs on shutdown but only if the `hwclock` daemon is enabled, hence for Arch systems which use systemd, this does not happen.

Note If the hwclock has been set again less than 24 hours after a previous set, the drift is not recalculated as `hwclock` considers the elapsed time period too short to accurately calculate the drift.

If the hardware clock keeps losing or gaining time in large increments, it is possible that an invalid drift has been recorded (but only applicable, if the hwclock daemon is running). This can happen if you have set the hardware clock time incorrectly or your time standard is not synchronized with a Windows or macOS install. The drift value can be removed by first removing the file `/etc/adjtime`, then setting the correct hardware clock and system clock time. You should then check if your time standard is correct.

Note If you wish to make use of the drift value stored in `/etc/adjtime` even when using systemd, (e.g. you cannot or do not want to use NTP), you must call `hwclock --adjust` on a regular basis, perhaps by creating a cron job.

The software clock is very accurate but like most clocks is not perfectly accurate and will drift as well. Though rarely, the system clock can lose accuracy if the kernel skips interrupts. There are some tools to improve software clock accuracy:

- See #Time synchronization.

Time synchronization

The Network Time Protocol (NTP) is a protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks.

Network Time Protocol (NTP)

For proper NTP support, as defined by the RFC, a client must be able to merge time from multiple servers, compensate for delay, and keep track of drift on the system (software) clock. The following are implementations of NTP available for Arch Linux:

- **Chrony** — A client and server that is roaming friendly and designed specifically for systems that are not online all the time. Converges faster and closer to reference than ntpd in most cases. Can also keep track of hardware clock (RTC) drift.

  https://chrony-project.org/ || chrony

- **Network Time Protocol daemon** — The reference implementation of the protocol.

  https://www.ntp.org/ || ntp

- **ntpd-rs** — A full-featured implementation of NTP with NTS support.

  https://docs.ntpd-rs.pendulum-project.org/ || ntpd-rs

- **NTPsec** — A fork of NTPd, focused on security. Works similarly, except a lot of old code is thrown out.

  https://ntpsec.org/ || ntpsec[AUR]

Simple Network Time Protocol (SNTP)

Anything that does less than a proper NTP node is considered Simple Network Time Protocol (SNTP). A basic SNTP client may simply fetch the time from a single server and set it immediately, without keeping track of long-term drifts. SNTP provides lower accuracy, but takes less resources. The accuracy is usually good enough for desktop users and embedded workloads, but unacceptable for NTP servers. The following implement SNTP:

- **ConnMan** — A lightweight network manager with SNTP support.

https://01.org/connman (waybackmachine) || connman

- **ntpclient** — A simple command-line SNTP client.

  http://doolittle.icarus.com/ntpclient/ || ntpclient[AUR]

- **OpenNTPD** — Part of the OpenBSD project and implements both an SNTP client and a server. No leap second support.

  https://www.openntpd.org/ || openntpd

- **sntp** — An SNTP client that comes with NTPd. It supersedes *ntpdate* and is recommended in non-server environments.

  https://www.ntp.org/ || ntp

- **systemd-timesyncd** — A simple SNTP daemon that only implements a client side, focusing only on querying time from one remote server. It should be more than appropriate for most installations.

  https://systemd.io/ || systemd

Per-user/session or temporary settings

For some use cases it may be useful to change the time settings without touching the global system values. For example to test applications relying on the time during development or adjusting the system time zone when logging into a server remotely from another zone.

To make an application "see" a different date/time than the system one, you can use the faketime(1) utility (from libfaketime).

If instead you want an application to "see" a different time zone than the system one, set the `TZ` environment variable, for example:

```
$ date && export TZ=":/usr/share/zoneinfo/Pacific/Fiji" && date
```

```
Tue Nov  1 14:34:51 CET 2016
Wed Nov  2 01:34:51 FJT 2016
```

This is different than just setting the time, as for example it allows to test the behavior of a program with positive or negative UTC offset values, or the effects of DST changes when developing on systems in a non-DST time zone.

Another use case is having different time zones set for different users of the same system: this can be accomplished by setting the `TZ` variable in the shell's configuration file, see Environment variables#Defining variables.

Tips and tricks

fake-hwclock

alarm-fake-hwclock designed especially for system without battery backed up RTC, it includes a systemd service which on shutdown saves the current time and on startup restores the saved time, thus avoiding strange time travel errors.

Install fake-hwclock-git[AUR], start/enable the service `fake-hwclock.service`.

Virtual PTP

Virtual machine guests may obtain time from the host machine using the PTP (Precision Time Protocol) `/dev/ptp0` interface. The interface is more accurate compared to using NTP over IP between the host and guest.

- On KVM machines, the `ptp_kvm` kernel module needs to be loaded to provide a virtual PTP device. See VM timekeeping: Using the PTP Hardware Clock on KVM.
- On Hyper-V machines, the guest integration should spawn a `/dev/ptp0` without additional configuration.

chrony and ntpd can each use the virtual-PTP device to sync the time between guest and host, by configuring the device as if it is a real PTP reference clock.

Troubleshooting

Clock shows a value that is neither UTC nor local time

This might be caused by a number of reasons. For example, if your hardware clock is running on local time, but `timedatectl` is set to assume it is in UTC, the result would be that your timezone's offset to UTC effectively gets applied twice, resulting in wrong values for your local time and UTC.

To force your clock to the correct time, and to also write the correct UTC to your hardware clock, follow these steps:

- Setup ntpd (enabling it as a service is not necessary).
- Set your time zone correctly.
- Run `ntpd -qg` to manually synchronize your clock with the network, ignoring large deviations between local UTC and network UTC.
- Run `hwclock --systohc` to write the current software UTC time to the hardware clock.

See also

- Linux Tips - Linux, Clocks, and Time
- An introduction to timekeeping in Linux VMs
- Sources for Time Zone and Daylight Saving Time Data for tzdata
- Time Scales
- Gentoo: System time

- Wikipedia:Time
