NAME

hwclock – time clocks utility

SYNOPSIS

hwclock [*function*] [*option*...]

DESCRIPTION

hwclock is an administration tool for the time clocks. It can: display the Hardware Clock time; set the Hardware Clock to a specified time; set the Hardware Clock from the System Clock; set the System Clock from the Hardware Clock; compensate for Hardware Clock drift; correct the System Clock timescale; set the kernel's timezone, NTP timescale, and epoch (Alpha only); and predict future Hardware Clock values based on its drift rate.

Since v2.26 important changes were made to the **−−hctosys** function and the **−−directisa** option, and a new option **−−update−drift** was added. See their respective descriptions below.

FUNCTIONS

The following functions are mutually exclusive, only one can be given at a time. If none is given, the default is **−−show**.

−a, −−adjust

Add or subtract time from the Hardware Clock to account for systematic drift since the last time the clock was set or adjusted. See the discussion below, under **The Adjust Function**.

−−getepoch

−−setepoch

These functions are for Alpha machines only, and are only available through the Linux kernel RTC driver.

They are used to read and set the kernel's Hardware Clock epoch value. Epoch is the number of years into AD to which a zero year value in the Hardware Clock refers. For example, if the machine's BIOS sets the year counter in the Hardware Clock to contain the number of full years since 1952, then the kernel's Hardware Clock epoch value must be 1952.

The **−−setepoch** function requires using the **−−epoch** option to specify the year. For example:

hwclock −−setepoch −−epoch=1952

The RTC driver attempts to guess the correct epoch value, so setting it may not be required.

This epoch value is used whenever **hwclock** reads or sets the Hardware Clock on an Alpha machine. For ISA machines the kernel uses the fixed Hardware Clock epoch of 1900.

−−predict

Predict what the Hardware Clock will read in the future based upon the time given by the **−−date** option and the information in */etc/adjtime*. This is useful, for example, to account for drift when setting a Hardware Clock wakeup (aka alarm). See **rtcwake**(8).

Do not use this function if the Hardware Clock is being modified by anything other than the current operating system's **hwclock** command, such as '11 minute mode' or from dual-booting another OS.

−r, **−−show**

−−get

Read the Hardware Clock and print its time to standard output in the **ISO 8601** format. The time shown is always in local time, even if you keep your Hardware Clock in UTC. See the **−−localtime** option.

Showing the Hardware Clock time is the default when no function is specified.

The **−−get** function also applies drift correction to the time read, based upon the information in

/etc/adjtime. Do not use this function if the Hardware Clock is being modified by anything other than the current operating system's **hwclock** command, such as '11 minute mode' or from dual-booting another OS.

−s, **−−hctosys**
Set the System Clock from the Hardware Clock. The time read from the Hardware Clock is compensated to account for systematic drift before using it to set the System Clock. See the discussion below, under **The Adjust Function**.

The System Clock must be kept in the UTC timescale for date-time applications to work correctly in conjunction with the timezone configured for the system. If the Hardware Clock is kept in local time then the time read from it must be shifted to the UTC timescale before using it to set the System Clock. The **−−hctosys** function does this based upon the information in the */etc/adjtime* file or the command line arguments **−−localtime** and **−−utc**. Note: no daylight saving adjustment is made. See the discussion below, under **LOCAL vs UTC**.

The kernel also keeps a timezone value, the **−−hctosys** function sets it to the timezone configured for the system. The system timezone is configured by the TZ environment variable or the */etc/localtime* file, as **tzset**(3) would interpret them. The obsolete tz_dsttime field of the kernel's timezone value is set to zero. (For details on what this field used to mean, see **settimeofday**(2).)

When used in a startup script, making the **−−hctosys** function the first caller of **settimeofday**(2) from boot, it will set the NTP '11 minute mode' timescale via the *persistent_clock_is_local* kernel variable. If the Hardware Clock's timescale configuration is changed then a reboot is required to inform the kernel. See the discussion below, under **Automatic Hardware Clock Synchronization by the Kernel**.

This is a good function to use in one of the system startup scripts before the file systems are mounted read/write.

This function should never be used on a running system. Jumping system time will cause problems, such as corrupted filesystem timestamps. Also, if something has changed the Hardware Clock, like NTP's '11 minute mode', then **−−hctosys** will set the time incorrectly by including drift compensation.

Drift compensation can be inhibited by setting the drift factor in */etc/adjtime* to zero. This setting will be persistent as long as the **−−update−drift** option is not used with **−−systohc** at shutdown (or anywhere else). Another way to inhibit this is by using the **−−noadjfile** option when calling the **−−hctosys** function. A third method is to delete the */etc/adjtime* file. **Hwclock** will then default to using the UTC timescale for the Hardware Clock. If the Hardware Clock is ticking local time it will need to be defined in the file. This can be done by calling **hwclock −−localtime −−adjust**; when the file is not present this command will not actually adjust the Clock, but it will create the file with local time configured, and a drift factor of zero.

A condition under which inhibiting **hwclock**'s drift correction may be desired is when dual-booting multiple operating systems. If while this instance of Linux is stopped, another OS changes the Hardware Clock's value, then when this instance is started again the drift correction applied will be incorrect.

For **hwclock**'s drift correction to work properly it is imperative that nothing changes the Hardware Clock while its Linux instance is not running.

−−set Set the Hardware Clock to the time given by the **−−date** option, and update the timestamps in */etc/adjtime*. With the **−−update-drift** option also (re)calculate the drift factor. Try it without the option if **−−set** fails. See **−−update-drift** below.

−−systz

This is an alternate to the **−−hctosys** function that does not read the Hardware Clock nor set the System Clock; consequently there is not any drift correction. It is intended to be used in a startup script on systems with kernels above version 2.6 where you know the System Clock has been set from the Hardware Clock by the kernel during boot.

It does the following things that are detailed above in the **−−hctosys** function:

- Corrects the System Clock timescale to UTC as needed. Only instead of accomplishing this by setting the System Clock, **hwclock** simply informs the kernel and it handles the change.
- Sets the kernel's NTP '11 minute mode' timescale.
- Sets the kernel's timezone.

The first two are only available on the first call of **settimeofday**(2) after boot. Consequently this option only makes sense when used in a startup script. If the Hardware Clocks timescale configuration is changed then a reboot would be required to inform the kernel.

−w, **−−systohc**

Set the Hardware Clock from the System Clock, and update the timestamps in */etc/adjtime*. With the **−−update-drift** option also (re)calculate the drift factor. Try it without the option if **−−systohc** fails. See **−−update-drift** below.

−V, **−−version**

Display version information and exit.

−h, **−−help**

Display help text and exit.

OPTIONS

−−adjfile=*filename*

Override the default */etc/adjtime* file path.

−−date=*date_string*

This option must be used with the **−−set** or **−−predict** functions, otherwise it is ignored.

hwclock −−set −−date='16:45'

hwclock −−predict −−date='2525-08-14 07:11:05'

The argument must be in local time, even if you keep your Hardware Clock in UTC. See the **−−localtime** option. Therefore, the argument should not include any timezone information. It also should not be a relative time like "+5 minutes", because **hwclock**'s precision depends upon correlation between the argument's value and when the enter key is pressed. Fractional seconds are silently dropped. This option is capable of understanding many time and date formats, but the previous parameters should be observed.

−−delay=*seconds*

This option allows to overwrite internally used delay when set clock time. The default is 0.5 (500ms) for rtc_cmos, for another RTC types the delay is 0. If RTC type is impossible to determine (from sysfs) then it defaults also to 0.5 to be backwardly compatible.

The 500ms default is based on commonly used MC146818A-compatible (x86) hardware clock. This Hardware Clock can only be set to any integer time plus one half second. The integer time is required because there is no interface to set or get a fractional second. The additional half second delay is because the Hardware Clock updates to the following second precisely 500 ms after setting the new time. Unfortunately, this behavior is hardware specific and in same cases another delay is required.

−D, **−−debug**

Use **−−verbose**. The **−−debug** option has been deprecated and may be repurposed or removed in a future release.

−−directisa
This option is meaningful for ISA compatible machines in the x86 and x86_64 family. For other machines, it has no effect. This option tells **hwclock** to use explicit I/O instructions to access the Hardware Clock. Without this option, **hwclock** will use the rtc device file, which it assumes to be driven by the Linux RTC device driver. As of v2.26 it will no longer automatically use directisa when the rtc driver is unavailable; this was causing an unsafe condition that could allow two processes to access the Hardware Clock at the same time. Direct hardware access from userspace should only be used for testing, troubleshooting, and as a last resort when all other methods fail. See the **−−rtc** option.

−−epoch=*year*
This option is required when using the **−−setepoch** function. The minimum *year* value is 1900. The maximum is system dependent (**ULONG_MAX - 1**).

−f, **−−rtc=***filename*
Override **hwclock**'s default rtc device file name. Otherwise it will use the first one found in this order:
/dev/rtc0
/dev/rtc
/dev/misc/rtc
For **IA-64:**
/dev/efirtc
/dev/misc/efirtc

−l, **−−localtime**
−u, **−−utc**
Indicate which timescale the Hardware Clock is set to.

The Hardware Clock may be configured to use either the UTC or the local timescale, but nothing in the clock itself says which alternative is being used. The **−−localtime** or **−−utc** options give this information to the **hwclock** command. If you specify the wrong one (or specify neither and take a wrong default), both setting and reading the Hardware Clock will be incorrect.

If you specify neither **−−utc** nor **−−localtime** then the one last given with a set function (**−−set**, **−−systohc**, or **−−adjust**), as recorded in */etc/adjtime*, will be used. If the adjtime file doesn't exist, the default is UTC.

Note: daylight saving time changes may be inconsistent when the Hardware Clock is kept in local time. See the discussion below, under **LOCAL vs UTC**.

−−noadjfile
Disable the facilities provided by */etc/adjtime*. **hwclock** will not read nor write to that file with this option. Either **−−utc** or **−−localtime** must be specified when using this option.

−−test Do not actually change anything on the system, that is, the Clocks or */etc/adjtime* (**−−verbose** is implicit with this option).

−−update−drift
Update the Hardware Clock's drift factor in */etc/adjtime*. It can only be used with **−−set** or **−−systohc**,

A minimum four hour period between settings is required. This is to avoid invalid calculations. The longer the period, the more precise the resulting drift factor will be.

This option was added in v2.26, because it is typical for systems to call **hwclock −−systohc** at shutdown; with the old behaviour this would automatically (re)calculate the drift factor which caused several problems:

- When using NTP with an '11 minute mode' kernel the drift factor would be clobbered to near zero.
- It would not allow the use of 'cold' drift correction. With most configurations using 'cold' drift will yield favorable results. Cold, means when the machine is turned off which can have a significant impact on the drift factor.
- (Re)calculating drift factor on every shutdown delivers suboptimal results. For example, if ephemeral conditions cause the machine to be abnormally hot the drift factor calculation would be out of range.
- Significantly increased system shutdown times (as of v2.31 when not using **−−update−drift** the RTC is not read).

Having **hwclock** calculate the drift factor is a good starting point, but for optimal results it will likely need to be adjusted by directly editing the */etc/adjtime* file. For most configurations once a machine's optimal drift factor is crafted it should not need to be changed. Therefore, the old behavior to automatically (re)calculate drift was changed and now requires this option to be used. See the discussion below, under **The Adjust Function**.

This option requires reading the Hardware Clock before setting it. If it cannot be read, then this option will cause the set functions to fail. This can happen, for example, if the Hardware Clock is corrupted by a power failure. In that case, the clock must first be set without this option. Despite it not working, the resulting drift correction factor would be invalid anyway.

−v, **−−verbose**
Display more details about what **hwclock** is doing internally.

NOTES

Clocks in a Linux System

There are two types of date-time clocks:

The Hardware Clock: This clock is an independent hardware device, with its own power domain (battery, capacitor, etc), that operates when the machine is powered off, or even unplugged.

On an ISA compatible system, this clock is specified as part of the ISA standard. A control program can read or set this clock only to a whole second, but it can also detect the edges of the 1 second clock ticks, so the clock actually has virtually infinite precision.

This clock is commonly called the hardware clock, the real time clock, the RTC, the BIOS clock, and the CMOS clock. Hardware Clock, in its capitalized form, was coined for use by **hwclock**. The Linux kernel also refers to it as the persistent clock.

Some non-ISA systems have a few real time clocks with only one of them having its own power domain. A very low power external I2C or SPI clock chip might be used with a backup battery as the hardware clock to initialize a more functional integrated real-time clock which is used for most other purposes.

The System Clock: This clock is part of the Linux kernel and is driven by a timer interrupt. (On an ISA machine, the timer interrupt is part of the ISA standard.) It has meaning only while Linux is running on the machine. The System Time is the number of seconds since 00:00:00 January 1, 1970 UTC (or more succinctly, the number of seconds since 1969 UTC). The System Time is not an integer, though. It has virtually infinite precision.

The System Time is the time that matters. The Hardware Clock's basic purpose is to keep time when Linux is not running so that the System Clock can be initialized from it at boot. Note that in DOS, for which ISA was designed, the Hardware Clock is the only real time clock.

It is important that the System Time not have any discontinuities such as would happen if you used the **date**(1) program to set it while the system is running. You can, however, do whatever you want to the Hardware Clock while the system is running, and the next time Linux starts up, it will do so with the adjusted time from the Hardware Clock. Note: currently this is not possible on most systems because **hwclock −−systohc** is called at shutdown.

The Linux kernel's timezone is set by **hwclock**. But don't be misled -- almost nobody cares what timezone the kernel thinks it is in. Instead, programs that care about the timezone (perhaps because they want to display a local time for you) almost always use a more traditional method of determining the timezone: They use the TZ environment variable or the */etc/localtime* file, as explained in the man page for **tzset**(3). However, some programs and fringe parts of the Linux kernel such as filesystems use the kernel's timezone value. An example is the vfat filesystem. If the kernel timezone value is wrong, the vfat filesystem will report and set the wrong timestamps on files. Another example is the kernel's NTP '11 minute mode'. If the kernel's timezone value and/or the *persistent_clock_is_local* variable are wrong, then the Hardware Clock will be set incorrectly by '11 minute mode'. See the discussion below, under **Automatic Hardware Clock Synchronization by the Kernel**.

hwclock sets the kernel's timezone to the value indicated by TZ or */etc/localtime* with the **−−hctosys** or **−−systz** functions.

The kernel's timezone value actually consists of two parts: 1) a field tz_minuteswest indicating how many minutes local time (not adjusted for DST) lags behind UTC, and 2) a field tz_dsttime indicating the type of Daylight Savings Time (DST) convention that is in effect in the locality at the present time. This second field is not used under Linux and is always zero. See also **settimeofday**(2).

Hardware Clock Access Methods

hwclock uses many different ways to get and set Hardware Clock values. The most normal way is to do I/O to the rtc device special file, which is presumed to be driven by the rtc device driver. Also, Linux systems using the rtc framework with udev, are capable of supporting multiple Hardware Clocks. This may bring about the need to override the default rtc device by specifying one with the **−−rtc** option.

However, this method is not always available as older systems do not have an rtc driver. On these systems, the method of accessing the Hardware Clock depends on the system hardware.

On an ISA compatible system, **hwclock** can directly access the "CMOS memory" registers that constitute the clock, by doing I/O to Ports 0x70 and 0x71. It does this with actual I/O instructions and consequently can only do it if running with superuser effective userid. This method may be used by specifying the **−−directisa** option.

This is a really poor method of accessing the clock, for all the reasons that userspace programs are generally not supposed to do direct I/O and disable interrupts. **hwclock** provides it for testing, troubleshooting, and because it may be the only method available on ISA systems which do not have a working rtc device driver.

The Adjust Function

The Hardware Clock is usually not very accurate. However, much of its inaccuracy is completely predictable - it gains or loses the same amount of time every day. This is called systematic drift. **hwclock**'s **−−adjust** function lets you apply systematic drift corrections to the Hardware Clock.

It works like this: **hwclock** keeps a file, */etc/adjtime*, that keeps some historical information. This is called the adjtime file.

Suppose you start with no adjtime file. You issue a **hwclock −−set** command to set the Hardware Clock to the true current time. **hwclock** creates the adjtime file and records in it the current time as the last time the clock was calibrated. Five days later, the clock has gained 10 seconds, so you issue a **hwclock −−set −−update−drift** command to set it back 10 seconds. **hwclock** updates the adjtime file to show the current time as the last time the clock was calibrated, and records 2 seconds per day as the systematic drift rate. 24 hours go by, and then you issue a **hwclock −−adjust** command. **hwclock** consults the adjtime file and sees that the clock gains 2 seconds per day when left alone and that it has been left alone for exactly one day. So it subtracts 2 seconds from the Hardware Clock. It then records the current time as the last time the clock was adjusted. Another 24 hours go by and you issue another **hwclock −−adjust**. **hwclock** does the same thing: subtracts 2 seconds and updates the adjtime file with the current time as the last time the clock was adjusted.

When you use the **−−update−drift** option with **−−set** or **−−systohc**, the systematic drift rate is (re)calculated by comparing the fully drift corrected current Hardware Clock time with the new set time, from that it

derives the 24 hour drift rate based on the last calibrated timestamp from the adjtime file. This updated drift factor is then saved in */etc/adjtime*.

A small amount of error creeps in when the Hardware Clock is set, so **--adjust** refrains from making any adjustment that is less than 1 second. Later on, when you request an adjustment again, the accumulated drift will be more than 1 second and **--adjust** will make the adjustment including any fractional amount.

hwclock --hctosys also uses the adjtime file data to compensate the value read from the Hardware Clock before using it to set the System Clock. It does not share the 1 second limitation of **--adjust**, and will correct sub-second drift values immediately. It does not change the Hardware Clock time nor the adjtime file. This may eliminate the need to use **--adjust**, unless something else on the system needs the Hardware Clock to be compensated.

The Adjtime File

While named for its historical purpose of controlling adjustments only, it actually contains other information used by **hwclock** from one invocation to the next.

The format of the adjtime file is, in ASCII:

Line 1: Three numbers, separated by blanks: 1) the systematic drift rate in seconds per day, floating point decimal; 2) the resulting number of seconds since 1969 UTC of most recent adjustment or calibration, decimal integer; 3) zero (for compatibility with **clock**(8)) as a decimal integer.

Line 2: One number: the resulting number of seconds since 1969 UTC of most recent calibration. Zero if there has been no calibration yet or it is known that any previous calibration is moot (for example, because the Hardware Clock has been found, since that calibration, not to contain a valid time). This is a decimal integer.

Line 3: "UTC" or "LOCAL". Tells whether the Hardware Clock is set to Coordinated Universal Time or local time. You can always override this value with options on the **hwclock** command line.

You can use an adjtime file that was previously used with the **clock**(8) program with **hwclock**.

Automatic Hardware Clock Synchronization by the Kernel

You should be aware of another way that the Hardware Clock is kept synchronized in some systems. The Linux kernel has a mode wherein it copies the System Time to the Hardware Clock every 11 minutes. This mode is a compile time option, so not all kernels will have this capability. This is a good mode to use when you are using something sophisticated like NTP to keep your System Clock synchronized. (NTP is a way to keep your System Time synchronized either to a time server somewhere on the network or to a radio clock hooked up to your system. See RFC 1305.)

If the kernel is compiled with the '11 minute mode' option it will be active when the kernel's clock discipline is in a synchronized state. When in this state, bit 6 (the bit that is set in the mask 0x0040) of the kernel's *time_status* variable is unset. This value is output as the 'status' line of the **adjtimex --print** or **ntptime** commands.

It takes an outside influence, like the NTP daemon to put the kernel's clock discipline into a synchronized state, and therefore turn on '11 minute mode'. It can be turned off by running anything that sets the System Clock the old fashioned way, including **hwclock --hctosys**. However, if the NTP daemon is still running, it will turn '11 minute mode' back on again the next time it synchronizes the System Clock.

If your system runs with '11 minute mode' on, it may need to use either **--hctosys** or **--systz** in a startup script, especially if the Hardware Clock is configured to use the local timescale. Unless the kernel is informed of what timescale the Hardware Clock is using, it may clobber it with the wrong one. The kernel uses UTC by default.

The first userspace command to set the System Clock informs the kernel what timescale the Hardware Clock is using. This happens via the *persistent_clock_is_local* kernel variable. If **--hctosys** or **--systz** is the first, it will set this variable according to the adjtime file or the appropriate command-line argument. Note that when using this capability and the Hardware Clock timescale configuration is changed, then a reboot is required to notify the kernel.

hwclock --adjust should not be used with NTP '11 minute mode'.

ISA Hardware Clock Century value

There is some sort of standard that defines CMOS memory Byte 50 on an ISA machine as an indicator of what century it is. **hwclock** does not use or set that byte because there are some machines that don't define the byte that way, and it really isn't necessary anyway, since the year-of-century does a good job of implying which century it is.

If you have a bona fide use for a CMOS century byte, contact the **hwclock** maintainer; an option may be appropriate.

Note that this section is only relevant when you are using the "direct ISA" method of accessing the Hardware Clock. ACPI provides a standard way to access century values, when they are supported by the hardware.

DATE-TIME CONFIGURATION

Keeping Time without External Synchronization

This discussion is based on the following conditions:

- Nothing is running that alters the date-time clocks, such as NTP daemon or a cron job."
- The system timezone is configured for the correct local time. See below, under **POSIX vs 'RIGHT'**.
- Early during startup the following are called, in this order:
  adjtimex −−tick *value* **−−frequency** *value*
  hwclock −−hctosys
- During shutdown the following is called:
  hwclock −−systohc

  ***** Systems without **adjtimex** may use **ntptime**.

Whether maintaining precision time with NTP daemon or not, it makes sense to configure the system to keep reasonably good date-time on its own.

The first step in making that happen is having a clear understanding of the big picture. There are two completely separate hardware devices running at their own speed and drifting away from the 'correct' time at their own rates. The methods and software for drift correction are different for each of them. However, most systems are configured to exchange values between these two clocks at startup and shutdown. Now the individual device's time keeping errors are transferred back and forth between each other. Attempt to configure drift correction for only one of them, and the other's drift will be overlaid upon it.

This problem can be avoided when configuring drift correction for the System Clock by simply not shutting down the machine. This, plus the fact that all of **hwclock**'s precision (including calculating drift factors) depends upon the System Clock's rate being correct, means that configuration of the System Clock should be done first.

The System Clock drift is corrected with the **adjtimex**(8) command's **−−tick** and **−−frequency** options. These two work together: tick is the coarse adjustment and frequency is the fine adjustment. (For systems that do not have an **adjtimex** package, **ntptime −f** *ppm* may be used instead.)

Some Linux distributions attempt to automatically calculate the System Clock drift with **adjtimex**'s compare operation. Trying to correct one drifting clock by using another drifting clock as a reference is akin to a dog trying to catch its own tail. Success may happen eventually, but great effort and frustration will likely precede it. This automation may yield an improvement over no configuration, but expecting optimum results would be in error. A better choice for manual configuration would be **adjtimex**'s **−−log** options.

It may be more effective to simply track the System Clock drift with **sntp**, or **date −Ins** and a precision timepiece, and then calculate the correction manually.

After setting the tick and frequency values, continue to test and refine the adjustments until the System Clock keeps good time. See **adjtimex**(8) for more information and the example demonstrating manual drift calculations.

Once the System Clock is ticking smoothly, move on to the Hardware Clock.

As a rule, cold drift will work best for most use cases. This should be true even for 24/7 machines whose

normal downtime consists of a reboot. In that case the drift factor value makes little difference. But on the rare occasion that the machine is shut down for an extended period, then cold drift should yield better results.

Steps to calculate cold drift:

1 **Ensure that NTP daemon will not be launched at startup.**

2 The *System Clock* time must be correct at shutdown!

3 Shut down the system.

4 Let an extended period pass without changing the Hardware Clock.

5 Start the system.

6 Immediately use **hwclock** to set the correct time, adding the **−−update−drift** option.

Note: if step 6 uses **−−systohc**, then the System Clock must be set correctly (step 6a) just before doing so.

Having **hwclock** calculate the drift factor is a good starting point, but for optimal results it will likely need to be adjusted by directly editing the */etc/adjtime* file. Continue to test and refine the drift factor until the Hardware Clock is corrected properly at startup. To check this, first make sure that the System Time is correct before shutdown and then use **sntp**, or **date −Ins** and a precision timepiece, immediately after startup.

LOCAL vs UTC

Keeping the Hardware Clock in a local timescale causes inconsistent daylight saving time results:

- If Linux is running during a daylight saving time change, the time written to the Hardware Clock will be adjusted for the change.
- If Linux is NOT running during a daylight saving time change, the time read from the Hardware Clock will NOT be adjusted for the change.

The Hardware Clock on an ISA compatible system keeps only a date and time, it has no concept of timezone nor daylight saving. Therefore, when **hwclock** is told that it is in local time, it assumes it is in the 'correct' local time and makes no adjustments to the time read from it.

Linux handles daylight saving time changes transparently only when the Hardware Clock is kept in the UTC timescale. Doing so is made easy for system administrators as **hwclock** uses local time for its output and as the argument to the **−−date** option.

POSIX systems, like Linux, are designed to have the System Clock operate in the UTC timescale. The Hardware Clock's purpose is to initialize the System Clock, so also keeping it in UTC makes sense.

Linux does, however, attempt to accommodate the Hardware Clock being in the local timescale. This is primarily for dual-booting with older versions of MS Windows. From Windows 7 on, the RealTimeIsUniversal registry key is supposed to be working properly so that its Hardware Clock can be kept in UTC.

POSIX vs 'RIGHT'

A discussion on date-time configuration would be incomplete without addressing timezones, this is mostly well covered by **tzset**(3). One area that seems to have no documentation is the 'right' directory of the Time Zone Database, sometimes called tz or zoneinfo.

There are two separate databases in the zoneinfo system, posix and 'right'. 'Right' (now named zoneinfo−leaps) includes leap seconds and posix does not. To use the 'right' database the System Clock must be set to (UTC + leap seconds), which is equivalent to (TAI − 10). This allows calculating the exact number of seconds between two dates that cross a leap second epoch. The System Clock is then converted to the correct civil time, including UTC, by using the 'right' timezone files which subtract the leap seconds. Note: this configuration is considered experimental and is known to have issues.

To configure a system to use a particular database all of the files located in its directory must be copied to the root of */usr/share/zoneinfo*. Files are never used directly from the posix or 'right' subdirectories, e.g., TZ='*right/Europe/Dublin*'. This habit was becoming so common that the upstream zoneinfo project restructured the system's file tree by moving the posix and 'right' subdirectories out of the zoneinfo directory and into sibling directories:

/usr/share/zoneinfo
/usr/share/zoneinfo−posix
/usr/share/zoneinfo−leaps

Unfortunately, some Linux distributions are changing it back to the old tree structure in their packages. So the problem of system administrators reaching into the 'right' subdirectory persists. This causes the system timezone to be configured to include leap seconds while the zoneinfo database is still configured to exclude them. Then when an application such as a World Clock needs the South_Pole timezone file; or an email MTA, or **hwclock** needs the UTC timezone file; they fetch it from the root of */usr/share/zoneinfo* , because that is what they are supposed to do. Those files exclude leap seconds, but the System Clock now includes them, causing an incorrect time conversion.

Attempting to mix and match files from these separate databases will not work, because they each require the System Clock to use a different timescale. The zoneinfo database must be configured to use either posix or 'right', as described above, or by assigning a database path to the **TZDIR** environment variable.

EXIT STATUS

One of the following exit values will be returned:

EXIT_SUCCESS ('0' on POSIX systems)
Successful program execution.

EXIT_FAILURE ('1' on POSIX systems)
The operation failed or the command syntax was not valid.

ENVIRONMENT

TZ If this variable is set its value takes precedence over the system configured timezone.

TZDIR
If this variable is set its value takes precedence over the system configured timezone database directory path.

FILES

/etc/adjtime
The configuration and state file for hwclock.

/etc/localtime
The system timezone file.

/usr/share/zoneinfo/
The system timezone database directory.

Device files **hwclock** may try for Hardware Clock access:
/dev/rtc0
/dev/rtc
/dev/misc/rtc
/dev/efirtc
/dev/misc/efirtc

SEE ALSO

date(1), **adjtimex**(8), **gettimeofday**(2), **settimeofday**(2), **crontab**(1), **tzset**(3)

AUTHORS

Written by Bryan Henderson, September 1996 (bryanh@giraffe-data.com), based on work done on the **clock**(8) program by Charles Hedrick, Rob Hooft, and Harald Koenig. See the source code for complete history and credits.

AVAILABILITY

The hwclock command is part of the util-linux package and is available from https://www.kernel.org/pub/linux/utils/util-linux/.