NAME

hwclock - query or set the hardware clock (RTC)

SYNOPSIS

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

DESCRIPTION

hwclock is a tool for accessing the Hardware Clock. You can display the current time, set the Hardware Clock to a specified time, set the Hardware Clock from the System Time, or set the System Time from the Hardware Clock.

You can also run **hwclock** periodically to add or subtract time from the Hardware Clock to compensate for systematic drift (where the clock consistently loses or gains time at a certain rate when left to run).

FUNCTIONS

You need exactly one of the following options to tell **hwclock** what function to perform:

-r, **--show**
Read the Hardware Clock and print the time on standard output. The time shown is always in local time, even if you keep your Hardware Clock in Coordinated Universal Time. See the **--utc** option. Showing the Hardware Clock time is the default when no function is specified.

--set Set the Hardware Clock to the time given by the **--date** option.

-s, **--hctosys**
Set the System Time from the Hardware Clock.

Also set the kernel's timezone value to the local timezone as indicated by the TZ environment variable and/or */usr/share/zoneinfo*, as tzset(3) would interpret them. The obsolete tz_dsttime field of the kernel's timezone value is set to DST_NONE. (For details on what this field used to mean, see settimeofday(2).)

This is a good option to use in one of the system startup scripts.

-w, **--systohc**
Set the Hardware Clock to the current System Time.

--systz
Set the kernel's timezone and reset the System Time based on the current timezone.

The system time is only reset on the first call after boot.

The local timezone is taken to be what is indicated by the TZ environment variable and/or */usr/share/zoneinfo*, as tzset(3) would interpret them. The obsolete tz_dsttime field of the kernel's timezone value is set to DST_NONE. (For details on what this field used to mean, see settimeofday(2).)

This is an alternate option to **--hctosys** that does not read the hardware clock, and may be used in system startup scripts for recent 2.6 kernels where you know the System Time contains the Hardware Clock time. If the Hardware Clock is already in UTC, it is not reset.

--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 discussion below.

--getepoch
Print the kernel's Hardware Clock epoch value to standard output. This is the number of years into AD to which a zero year value in the Hardware Clock refers. For example, if you are using the convention that the year counter in your Hardware Clock contains the

number of full years since 1952, then the kernel's Hardware Clock epoch value must be 1952.

This epoch value is used whenever **hwclock** reads or sets the Hardware Clock.

--setepoch
Set the kernel's Hardware Clock epoch value to the value specified by the **--epoch** option. See the **--getepoch** option for details.

--predict
Predict what the RTC will read at time given by the **--date** option based on the adjtime file. This is useful for example if you need to set an RTC wakeup time to distant future and want to account for the RTC drift.

-c, **--compare**
Periodically compare the Hardware Clock to the System Time and output the difference every 10 seconds. This will also print the frequency offset and tick.

-h, **--help**
Display help text and exit.

-V, **--version**
Display version information and exit.

OPTIONS

The first two options apply to just a few specific functions, the others apply to most functions.

--date=*date_string*
You need this option if you specify the **--set** or **--predict** functions, otherwise it is ignored. It specifies the time to which to set the Hardware Clock, or the time for which to predict the Hardware Clock reading. The value of this option is an argument to the date(1) program. For example:

hwclock --set --date=2011-08-14 16:45:05

The argument must be in local time, even if you keep your Hardware Clock in Coordinated Universal time. See the **--utc** option.

--epoch=*year*
Specifies the year which is the beginning of the Hardware Clock's epoch, that is the number of years into AD to which a zero value in the Hardware Clock's year counter refers. It is used together with the **--setepoch** option to set the kernel's idea of the epoch of the Hardware Clock, or otherwise to specify the epoch for use with direct ISA access.

For example, on a Digital Unix machine:

hwclock --setepoch --epoch=1952

-u, **--utc**

--localtime
Indicates that the Hardware Clock is kept in Coordinated Universal Time or local time, respectively. It is your choice whether to keep your clock in UTC or local time, but nothing in the clock tells which you've chosen. So this option is how you give that information to **hwclock**.

If you specify the wrong one of these options (or specify neither and take a wrong default), both setting and querying of the Hardware Clock will be messed up.

If you specify neither **--utc** nor **--localtime**, the default is whichever was specified the last time **hwclock** was used to set the clock (i.e. **h wclock** was successfully run with the **--set**, **--systohc**, or **--adjust** options), as recorded in the adjtime file. If the adjtime file doesn't exist, the default is UTC time.

--noadjfile
Disables 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.

--adjfile=*filename*
Overrides the default /etc/adjtime.

-f, **--rtc=***filename*
Overrides the default /dev file name, which is */dev/rtc* on many platforms but may be */dev/rtc0*, */dev/rtc1*, and so on.

--directisa
This option is meaningful only on an ISA machine or an Alpha (which implements enough of ISA to be, roughly speaking, an ISA machine for **hwclock**'s purposes). 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,**h wclock** will try to use the /dev/rtc device (which it assumes to be driven by the RTC device driver). If it is unable to open the device (for reading), it will use the explicit I/O instructions anyway.

--badyear
Indicates that the Hardware Clock is incapable of storing years outside the range 1994-1999. There is a problem in some BIOSes (almost all Award BIOSes made between 4/26/94 and 5/31/95) wherein they are unable to deal with years after 1999. If one attempts to set the year-of-century value to something less than 94 (or 95 in some cases), the value that actually gets set is 94 (or 95). Thus, if you have one of these machines, **hwclock** cannot set the year after 1999 and cannot use the value of the clock as the true time in the normal way.

To compensate for this (without your getting a BIOS update, which would definitely be preferable), always use **--badyear** if you have one of these machines. When **hwclock** knows it's working with a brain-damaged clock, it ignores the year part of the Hardware Clock value and instead tries to guess the year based on the last calibrated date in the adjtime file, by assuming that date is within the past year. For this to work, you had better do a **hwclock --set** or **hwclock --systohc** at least once a year!

Though **hwclock** ignores the year value when it reads the Hardware Clock, it sets the year value when it sets the clock. It sets it to 1995, 1996, 1997, or 1998, whichever one has the same position in the leap year cycle as the true year. That way, the Hardware Clock inserts leap days where they belong. Again, if you let the Hardware Clock run for more than a year without setting it, this scheme could be defeated and you could end up losing a day.

hwclock warns you that you probably need **--badyear** whenever it finds your Hardware Clock set to 1994 or 1995.

--srm This option is equivalent to **--epoch=1900** and is used to specify the most common epoch on Alphas with SRM console.

--arc This option is equivalent to **--epoch=1980** and is used to specify the most common epoch on Alphas with ARC console (but Ruffians have epoch 1900).

--jensen

--funky-toy

These two options specify what kind of Alpha machine you have. They are invalid if you don't have an Alpha and are usually unnecessary if you do, because **hwclock** should be able to determine by itself what it's running on, at least when */proc* is mounted. (If you find you need one of these options to make **hwclock** work, contact the maintainer to see if the program can be improved to detect your system automatically. Output of 'hwclock --debug' and 'cat /proc/cpuinfo' may be of interest.)

Option **--jensen** means you are running on a Jensen model. And **--funky-toy** means that on your machine one has to use the UF bit instead of the UIP bit in the Hardware Clock to detect a time transition. Toy in the option name refers to the Time Of Year facility of the machine.

--test Do everything except actually updating the Hardware Clock or anything else. This is useful, especially in conjunction with **--debug**, in learning about **hwclock**.

--debug

Display a lot of information about what **hwclock** is doing internally. Some of its function is complex and this output can help you understand how the program works.

NOTES

Clocks in a Linux System

There are two main clocks in a Linux system:

The Hardware Clock: This is a clock that runs independently of any control program running in the CPU and even when the machine is powered off.

On an ISA system, this clock is specified as part of the ISA standard. The control program can read or set this clock to a whole second, but the control program 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** because all of the other names are inappropriate to the point of being misleading.

So for example, 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 Time: This is the time kept by a clock inside the Linux kernel and 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). 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 in a Linux system is to keep time when Linux is not running. You initialize the System Time to the time from the Hardware Clock when Linux starts up, and then never use the Hardware Clock again. 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(1L) 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.

A Linux kernel maintains a concept of a local timezone for the system. 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 and/or the */usr/share/zoneinfo* directory, 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 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.

hwclock sets the kernel timezone to the value indicated by TZ and/or */usr/share/zoneinfo* when you set the System Time using the **--hctosys** option.

The 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).)

Users access and setuid

Sometimes, you need to install **hwclock** setuid root. If you want users other than the superuser to be able to display the clock value using the direct ISA I/O method, install it setuid root. If you have the /dev/rtc interface on your system or are on a non-ISA system, there's probably no need for users to use the direct ISA I/O method, so don't bother.

In any case, hwclock will not allow you to set anything unless you have the superuser real uid. (This is restriction is not necessary if you haven't installed setuid root, but it's there for now).

How hwclock Accesses the Hardware Clock

hwclock uses many different ways to get and set Hardware Clock values. The most normal way is to do I/O to the device special file /dev/rtc, which is presumed to be driven by the rtc device driver. However, this method is not always available. For one thing, the rtc driver is a relatively recent addition to Linux. Older systems don't have it. Also, though there are versions of the rtc driver that work on DEC Alphas, there appear to be plenty of Alphas on which the rtc driver does not work (a common symptom is hwclock hanging). Moreover, recent Linux systems have more generic support for RTCs, even systems that have more than one, so you might need to override the default by specifying */dev/rtc0* or */dev/rtc1* instead.

On older systems, the method of accessing the Hardware Clock depends on the system hardware.

On an ISA 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. (In the case of a Jensen Alpha, there is no way for **hwclock** to execute those I/O instructions, and so it uses instead the /dev/port device special file, which provides almost as low-level an interface to the I/O subsystem).

This is a really poor method of accessing the clock, for all the reasons that user space programs are generally not supposed to do direct I/O and disable interrupts. Hwclock provides it because it is the only method available on ISA and Alpha systems which don't have working rtc device drivers available.

On an m68k system, **hwclock** can access the clock via the console driver, via the device special file /dev/tty1.

hwclock tries to use /dev/rtc. If it is compiled for a kernel that doesn't have that function or it is unable to open /dev/rtc (or the alternative special file you've defined on the command line) **hwclock** will fall back to another method, if available. On an ISA or Alpha machine, you can force **hwclock** to use the direct manipulation of the CMOS registers without even trying */dev/rtc* by specifying the **--directisa** option.

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 make systematic corrections to correct the systematic drift.

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. **Hw clock** creates the adjtime file and records in it the current time as the last time the clock was calibrated. 5 days later, the clock has gained 10 seconds, so you issue another *hwclock --set* command to set it back 10 seconds. **Hw clock** 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 goes 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.

Every time you calibrate (set) the clock (using *--set* or *--systohc*), **hwclock** recalculates the systematic drift rate based on how long it has been since the last calibration, how long it has been since the last adjustment, what drift rate was assumed in any intervening adjustments, and the amount by which the clock is presently off.

A small amount of error creeps in any time **hwclock** sets the clock, so it refrains from making an adjustment that would be less than 1 second. Later on, when you request an adjustment again, the accumulated drift will be more than a second and **hwclock** will do the adjustment then.

It is good to do a *hwclock --adjust* just before the *hwclock --hctosys* at system startup time, and maybe periodically while the system is running via cron.

The adjtime file, while named for its historical purpose of controlling adjustments only, actually contains other information for use by hwclock in remembering information from one invocation to the next.

The format of the adjtime file is, in ASCII:

Line 1: 3 numbers, separated by blanks: 1) systematic drift rate in seconds per day, floating point decimal; 2) 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: 1 number: 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 is a good mode to use when you are using something sophisticated like ntp to keep your System Time 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).

This mode (we'll call it 11 minute mode) is off until something turns it on. The ntp daemon xntpd is one thing that turns it on. You can turn it off by running anything, including *hwclock --hctosys*, that sets the System Time the old fashioned way.

If your system runs with 11 minute mode on, don't use *hwclock --adjust* or *hwclock --hctosys.* You'll just make a mess. It is acceptable to use a *hwclock --hctosys* at startup time to get a reasonable System Time until your system is able to set the System Time from the external source and start 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. **h wclock** 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.

ENVIRONMENT VARIABLES

TZ

FILES

/etc/adjtime */usr/share/zoneinfo/* (*/usr/lib/zoneinfo* on old systems) */dev/rtc* */dev/rtc0* */dev/port* */dev/tty1* */proc/cpuinfo*

SEE ALSO

date(1), 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* 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 ftp://ftp.kernel.org/pub/linux/utils/util-linux/.