The Clock Mini−HOWTO

Table of Contents

The Clock Mini−HOWTO

Ron Bean, rbean@execpc.com

v2.1, November 2000

How to set and keep your computer's clock on time.

1. Introduction

1.1 Does Anybody Really Know What Time It Is?

The Real−Time−Clock (RTC) chips used on PC motherboards are notoriously inaccurate, usually gaining or losing the same amount of time each day. Linux provides a simple way to correct for this in software, which

can make the clock **very** accurate, even without an external time source. But most people don't know how to set it up, for several reasons:

- It's not mentioned in most of the general documentation on how to set up linux, and it can't be set up automatically (unless you have an external time source), so the default is not to use it.
- If you type "`man clock`" you may get the man page for `clock(3)`, which is not what you want. Try "`man 8 clock`" or "`man 8 hwclock`" (some distributions search the man pages in numerical order if you don't give a section number, others search in the order specified in `/etc/man.config`).
- Most people don't seem to care what time it is anyway.
- Those few who do care often want to sync the system clock to an external time source, such as a network time server or radio clock. This makes the accuracy of the RTC (mostly) irrelevant.

This mini−HOWTO describes the low−tech approach (which can be very accurate by itself), and provides pointers to several more sophisticated options. In most cases the documentation is well written, so I'm not going to repeat that information here.

Previous versions included detailed instructions for the old `clock(8)` program for anyone still running an older system, but I've dropped that section because most distributions now use `hwclock(8)` instead, which has much better documentation. If you still want a copy of the `clock(8)` instructions I can email them to you, but read the section on `hwclock(8)` first.

Note

> You must be logged in as "**root**" to run any program that affects the RTC or the system time, which includes most of the programs described here. If you normally use a graphical interface for everything, you may also need to learn some basic unix shell commands.

Note

> If you run more than one OS on your machine, you should only let one of them set the RTC, so they don't confuse each other. The exception is the twice−a−year adjustment for Daylight Saving(s) Time (see the section on DST for details).

If you run a dual−boot system that spends a lot of time running Windows, you may want to check out some of the clock software available for that OS instead. Follow the links on the NTP website at http://www.eecis.udel.edu/~ntp/software.html. Many of the radio clocks mentioned here include software for Windows.

1.2 Where to Find Stuff: "The Usual Places"

In some places I've mentioned that software can be downloaded from "the usual places", which means any place you could download a complete Linux system if you didn't get it on a CD−ROM. In the old days that meant the ftp archive at sunsite.unc.edu, and various mirror sites around the world. That site has been renamed http://metalab.unc.edu/linux/ (since Sun no longer sponsors it). Some distributions also have their own websites, which may include a lot of this stuff.

I assume most people get Linux on CD these days, and those CDs often include software that is not part of the default installation, so you may already have some of the programs mentioned here without knowing it.

The latest version of this mini−HOWTO can be found at the home of the Linux Documentation Project, which is currently http://www.linuxdoc.org/ (and is also reachable from the metalab site mentioned above). I think all the old links are now forwarded to this one.

All HOWTOs are written in SGML and converted to various other formats by standardized conversion programs. Most people seem to want the HTML version, which is at http://www.linuxdoc.org/HOWTO/mini/Clock.html. Revision history can be found as comments in the SGML source. Most Linux distributions install a complete set of HOWTO's in `/usr/doc/HOWTO/` and `/usr/doc/HOWTO/mini`.

1.3 Acknowledgements

This mini−HOWTO has been greatly improved thanks to various people who have sent me email since the first version in 1996. In some cases they wrote with questions but ended up giving me as much information as I gave them. Unfortunately I haven't compiled a list of names (maybe next time). You know who you are `:−)`.

---

2. How Linux Keeps Track of Time

2.1 Basic Strategies

A Linux system actually has two clocks: One is the battery powered "Real Time Clock" (also known as the "RTC", "CMOS clock", or "Hardware clock") which keeps track of time when the system is turned off but is not used when the system is running. The other is the "system clock" (sometimes called the "kernel clock" or "software clock") which is a software counter based on the timer interrupt. It does not exist when the system is not running, so it has to be initialized from the RTC (or some other time source) at boot time. References to "the clock" in the `ntpd` documentation refer to the system clock, not the RTC.

The two clocks will drift at different rates, so they will gradually drift apart from each other, and also away from the "real" time. The simplest way to keep them on time is to measure their drift rates and apply correction factors in software. Since the RTC is only used when the system is not running, the correction factor is applied when the clock is read at boot time, using `clock(8)` or `hwclock(8)`. The system clock is corrected by adjusting the rate at which the system time is advanced with each timer interrupt, using `adjtimex(8)`.

A crude alternative to `adjtimex(8)` is to have `chron` run `clock(8)` or `hwclock(8)` periodically to sync the system time to the (corrected) RTC. This was recommended in the `clock(8)` man page, and it works if you do it often enough that you don't cause large "jumps" in the system time, but `adjtimex(8)` is a more elegant solution. Some applications may complain if the time jumps backwards.

The next step up in accuracy is to use a program like `ntpd` to read the time periodically from a network time server or radio clock, and continuously adjust the rate of the system clock so that the times always match, without causing sudden "jumps" in the system time. If you always have a network connection at boot time, you can ignore the RTC completely and use `ntpdate` (which comes with the `ntpd` package) to initialize the system clock from a time server−− either a local server on a LAN, or a remote server on the internet. But if you sometimes don't have a network connection, or if you need the time to be accurate during the boot sequence before the network is active, then you need to maintain the time in the RTC as well.

2.2 Potential Conflicts

It might seem obvious that if you're using a program like `ntpd`, you would want to sync the RTC to the (corrected) system clock. But this turns out to be a bad idea if the system is going to stay shut down longer than a few minutes, because it interferes with the programs that apply the correction factor to the RTC at boot time.

If the system runs 24/7 and is always rebooted immediately whenever it's shut down, then you can just set the RTC from the system clock right before you reboot. The RTC won't drift enough to make a difference in the time it takes to reboot, so you don't need to know its drift rate.

Of course the system may go down unexpectedly, so some versions of the kernel sync the RTC to the system clock every 11 minutes if the system clock has been adjusted by another program. The RTC won't drift enough in 11 minutes to make any difference, but if the system is down long enough for the RTC to drift significantly, then you have a problem: the programs that apply the drift correction to the RTC need to know **exactly** when it was last reset, and the kernel doesn't record that information anywhere.

Some unix "traditionalists" might wonder why anyone would run a linux system less than 24/7, but some of us run dual−boot systems with another OS running some of the time, or run Linux on laptops that have to be shut down to conserve battery power when they're not being used. Other people just don't like to leave machines running unattended for long periods of time (even though we've heard all the arguments in favor of it). So the "every 11 minutes" feature becomes a bug.

This "feature/bug" appears to behave differently in different versions of the kernel (and possibly in different versions of `xntpd` and `ntpd` as well), so if you're running both `ntpd` and `hwclock` you may need to test your system to see what it actually does. If you can't keep the kernel from resetting the RTC, you might have to run without a correction factor on the RTC.

The part of the kernel that controls this can be found in `/usr/src/linux−2.0.34/arch/i386/kernel/time.c` (where the version number in the path will be the version of the kernel you're running). If the variable `time_status` is set to `TIME_OK` then the kernel will write the system time to the RTC every 11 minutes, otherwise it leaves the RTC alone. Calls to `adjtimex(2)` (as used by `ntpd` and `timed`, for example) may turn this on. Calls to `settimeofday(2)` will set `time_status` to `TIME_UNSYNC`, which tells the kernel not to adjust the RTC. I have not found any real documentation on this.

I've heard reports that some versions of the kernel may have problems with "sleep modes" that shut down the CPU to save energy. The best solution is to keep your kernel up to date, and refer any problems to the people who maintain the kernel.

If you get bizarre results from the RTC you may have a hardware problem. Some RTC chips include a lithium battery that can run down, and some motherboards have an option for an external battery (be sure the jumper is set correctly). The same battery maintains the CMOS RAM, but the clock takes more power and is likely to fail first. Bizarre results from the system clock may mean there is a problem with interrupts.

2.3 Should the RTC use Local Time or UTC, and What About DST?

The Linux "system clock" actually just counts the number of seconds past Jan 1, 1970, and is always in UTC (or GMT, which is technically different but close enough that casual users tend to use both terms interchangeably). UTC does not change as DST comes and goes−− what changes is the *conversion* between UTC and local time. The translation to local time is done by library functions that are linked into the application programs.

This has two consequences: First, any application that needs to know the local time also needs to know what time zone you're in, and whether DST is in effect or not (see the next section for more on time zones). Second, there is no provision in the kernel to change either the system clock or the RTC as DST comes and goes, because UTC doesn't change. Therefore, machines that only run Linux should have the RTC set to UTC, not local time.

However, many people run dual−boot systems with other OS's that expect the RTC to contain the local time, so `hwclock` needs to know whether your RTC is in local time or UTC, which it then converts to seconds past Jan 1, 1970 (UTC). This still does not provide for seasonal changes to the RTC, so the change must be made by the other OS (this is the one exception to the rule against letting more than one program change the time in the RTC).

Unfortunately, there are no flags in the RTC or the CMOS RAM to indicate standard time vs DST, so each OS stores this information someplace where the other OS's can't find it. This means that `hwclock` must assume that the RTC always contains the correct local time, even if the other OS has not been run since the most recent seasonal time change.

If Linux is running when the seasonal time change occurs, the system clock is unaffected and applications will make the correct conversion. But if linux has to be rebooted for any reason, the system clock will be set to the time in the RTC, which will be off by one hour until the other OS (usually Windows) has a chance to run.

There is no way around this, but Linux doesn't crash very often, so the most likely reason to reboot on a dual−boot system is to run the other OS anyway. But beware if you're one of those people who shuts down Linux whenever you won't be using it for a while−− if you haven't had a chance to run the other OS since the last time change, the RTC will be off by an hour until you do.

Some other documents have stated that setting the RTC to UTC allows Linux to take care of DST properly. This is not really wrong, but it doesn't tell the whole story−− as long as you don't reboot, it does not matter which time is in the RTC (or even if the RTC's battery dies). Linux will maintain the correct time either way, until the next reboot. In theory, if you only reboot once a year (which is not unreasonable for Linux), DST could come and go and you'd never notice that the RTC had been wrong for several months, because the system clock would have stayed correct all along. But since you can't predict when you'll want to reboot, it's better to have the RTC set to UTC if you're not running another OS that requires local time.

The Dallas Semiconductor RTC chip (which is a drop−in replacement for the Motorola chip used in the IBM AT and clones) actually has the ability to do the DST conversion by itself, but this feature is not used because the changeover dates are hard−wired into the chip and can't be changed. Current versions change on the first Sunday in April and the last Sunday in October, but earlier versions used different dates (and obviously this doesn't work in countries that use other dates). Also, the RTC is often integrated into the motherboard's "chipset" (rather than being a separate chip) and I don't know if they all have this ability.

2.4 How Linux keeps Track of Time Zones

You probably set your time zone correctly when you installed Linux. But if you have to change it for some reason, or if the local laws regarding DST have changed (as they do frequently in some countries), then you'll need to know how to change it. If your system time is off by some exact number of hours, you may have a time zone problem (or a DST problem).

Time zone and DST information is stored in `/usr/share/zoneinfo` (or `/usr/lib/zoneinfo` on older systems). The local time zone is determined by a symbolic link from `/etc/localtime` to one of these files. The way to change your timezone is to change the link. If your local DST dates have changed, you'll have to edit the file.

You can also use the `TZ` environment variable to change the current time zone, which is handy of you're logged in remotely to a machine in another time zone. Also see the man pages for `tzset` and `tzfile`.

This is nicely summarized at http://www.linuxsa.org.au/tips/time.html

2.5 The Bottom Line

If you don't need sub−second accuracy, `hwclock(8)` and `adjtimex(8)` may be all you need. It's easy to get enthused about time servers and radio clocks and so on, but I ran the old `clock(8)` program for years with excellent results. On the other hand, if you have several machines on a LAN it can be handy (and sometimes essential) to have them automatically sync their clocks to each other. And the other stuff can be fun to play with even if you don't really need it.

On machines that only run Linux, set the RTC to UTC (or GMT). On dual−boot systems that require local time in the RTC, be aware that if you have to reboot Linux after the seasonal time change, the clock may be temporarily off by one hour, until you have a chance to run the other OS. If you run more than two OS's, be sure only one of them is trying to adjust for DST.

---

3. Software

3.1 Clock(8) and Hwclock(8)

All linux distributions install either the old `clock(8)` or the newer `hwclock(8)`, but without a correction factor. Some may also install `adjtimex(8)`, or they may include it on the CD as optional (or you can download it from the usual Linux archive sites). Some distributions also include a graphical clock setting program that runs in an X−window, but those are designed for interactive use, and the system will still install `clock(8)` or `hwclock(8)` for use in the startup scripts.

`Clock(8)` requires you to calculate the correction factor by hand, but `hwclock(8)` calculates it automatically whenever you use it to reset the RTC (using another program to set the RTC will interfere with the drift correction, so always use the same program if you're using a correction factor). If you have an older system that still uses `clock(8)` and you want to upgrade, you can find `hwclock(8)` in the "`util−linux`" package, version 2.7 or later. See the man page for more information.

Note

> The man page for `hwclock(8)` may be called "`clock`" for backward compatibility, so try both names. `Hwclock(8)` will respond to commands written for `clock(8)`, but the result may not be the same−− in particular, "`hwclock −a`" is not quite the same as "`clock −a`", so if you're upgrading to `hwclock` I'd suggest replacing all references to "`clock`" in your startup scripts to use `hwclock`'s native commands instead.

The startup scripts vary from one distribution to another, so you may have to search a bit to find where it sets the clock. Typical locations are `/etc/rc.local`, `/etc/rc.d/rc.sysinit`, `/etc/rc.d/boot`, or some similar place.

The correction factor for the RTC is stored in `/etc/adjtime`. Red Hat has a script in `/etc/sysconfig/clock` that controls the options to `hwclock`.

When you're setting the clock to determine the drift rate, keep in mind that your local telephone time announcement may or may not be accurate. If you don't have a shortwave radio or GPS receiver, you can hear the audio feed from WWV by calling (303)499−7111 (this is a toll call to Boulder, Colorado). It will cut you off after three minutes, but that should be long enough to set the clock. USNO and Canada's CHU also have telephone time services, but I prefer WWV's because there is more time between the announcement and the "beep". You can also get the time from a network time server using the `ntpdate` program that comes with `ntpd`, and there's a javaclock at www.time.gov.

In any case, what you're setting is the system clock, not the RTC (see the man page for the `date` command for the formats to use). Then use `hwclock` to set the RTC and calculate the drift rate. If you're doing this by hand, you should be able to set it within a second or two, and get a reasonable approximation of the drift rate after a few weeks. Then you can run `adjtimex(8)` to fine−tune the system clock.

3.2 Adjtimex(8)

`Adjtimex(8)` allows the user to adjust the kernel's time variables, and therefore change the speed of the system clock (you must be logged in as "**root**" to do this). It is cleverly designed to compare the system clock to the RTC using the same correction factor used by `clock(8)` or `hwclock(8)`, as stored in `/etc/adjtime`. So, once you've established the drift rate of the RTC, it's fairly simple to correct the system clock as well. Once you've got it running at the right speed, you can add a line to your startup scripts to set the correct kernel variables at boot time. Since `adjtimex(8)` was designed to work with `clock(8)` or `hwclock(8)`, it includes a work−around for the "every 11 minutes" bug.

After you've installed `adjtimex(8)` you can get more information on setting it up by typing "`man 8 adjtimex`" (there is also an `adjtimex(2)` man page, which is not what you want) and by reading the `README` file in `/usr/doc/adjtimex−1.3/README` (where the version number in the path will be the current version of `adjtimex(8)`).

3.3 Xntpd and ntpd: the Network Time Protocol

`Xntpd` (NTPv3) has been replaced by `ntpd` (NTPv4); the earlier version is no longer being maintained.

`Ntpd` is the standard program for synchronizing clocks across a network, and it comes with a list of public time servers you can connect to. It can be a little more complicated to set up than the other programs described here, but if you're interested in this kind of thing I highly recommend that you take a look at it anyway.

The "home base" for information on ntpd is the NTP website at http://www.eecis.udel.edu/~ntp/ which also includes links to all kinds of interesting time−related stuff (including software for other OS's). Some linux distributions include ntpd on the CD. There is a list of public time servers at http://www.eecis.udel.edu/~mills/ntp/clock2.html.

A relatively new feature in ntpd is a "burst mode" which is designed for machines that have only intermittent dial−up access to the internet.

Ntpd includes drivers for quite a few radio clocks (although some appear to be better supported than others). Most radio clocks are designed for commercial use and cost thousands of dollars, but there are some cheaper alternatives (discussed in later sections). In the past most were WWV or WWVB receivers, but now most of them seem to be GPS receivers. NIST has a PDF file that lists manufacturers of radio clocks on their website at http://www.boulder.nist.gov/timefreq/links.htm (near the bottom of the page). The NTP website also includes many links to manufacturers of radio clocks at http://www.eecis.udel.edu/~ntp/hardware.htm and http://www.eecis.udel.edu/~mills/ntp/refclock.htm. Either list may or may not be up to date at any given time :−). The list of drivers for ntpd is at http://www.eecis.udel.edu/~ntp/ntp_spool/html/refclock.htm.

Ntpd also includes drivers for several dial−up time services. These are all long−distance (toll) calls, so be sure to calculate the effect on your phone bill before using them.

3.4 Chrony

Xntpd was originally written for machines that have a full−time connection to a network time server or radio clock. In theory it can also be used with machines that are only connected intermittently, but Richard Curnow couldn't get it to work the way he wanted it to, so he wrote "chrony" as an alternative for those of us who only have network access when we're dialed in to an ISP (this is the same problem that ntpd's new "burst mode" was designed to solve). The current version of chrony includes drift correction for the RTC, for machines that are turned off for long periods of time.

You can get more information from Richard Curnow's website at http://www.rrbcurnow.freeuk.com/chrony or http://go.to/chrony. There are also two chrony mailing lists, one for announcements and one for discussion by users. For information send email to chrony−users−subscribe@egroups.com or chrony−announce−subscribe@egroups.com

Chrony is normally distributed as source code only, but Debian has been including a binary in their "unstable" collection. The source file is also available at the usual Linux archive sites.

3.5 Clockspeed

Another option is the clockspeed program by DJ Bernstein. It gets the time from a network time server and simply resets the system clock every three seconds. It can also be used to synchronize several machines on a LAN.

I've sometimes had trouble reaching his website at http://Cr.yp.to/clockspeed.html, so if you get a DNS error try again on another day. I'll try to update this section if I get some better information.

4. Radio Clocks

4.1 CHU and the "Gadget Box"

CHU, the Canadian shortwave time station near Ottawa, is similar to WWV in the US but with one important difference: in addition to announcing the time in both French and English, it also broadcasts the current time once per minute using the old "Bell 103" (300 baud) modem tones. These tones are very easy to decode, and Bill Rossi realised that you don't even need a modem−− just a shortwave radio and a sound card. If you're able to receive the signal from CHU, this may be the cheapest radio clock available. Shortwave reception varies throughout the day, but Bill claims that by changing frequencies twice a day (morning and evening) he gets almost 24−hour coverage. CHU broadcasts on 3.33, 7.335, and 14.670 MHz.

For more information see Bill Rossi's website at http://www.rossi.com/chu/. The source file is also available at the usual Linux archive sites. For information on CHU's time services see http://www.nrc.ca/inms/time/ctse.html.

The NTP website has plans for a "gadget box" that decodes the CHU time broadcast using an inexpensive 300 baud modem chip and any shortwave radio, at http://www.eecis.udel.edu/~ntp/ntp_spool/html/gadget.htm. The plans include a Postscript image of a 2−sided custom printed circuit board, but you have to make the board yourself (or find someone who can make it for you).

`Ntpd` includes a driver (type 7) for CHU receivers, which works either with modems like the "`gadget box`", or by feeding the audio directly into the mic input of a Sun SPARCstation (or any other machine with "compatible audio drivers").

4.2 WWV and the "Most Accurate Clock"

You may have heard about Heathkit's "Most Accurate Clock", which received and decoded the time signal from WWV and had an optional serial port for connecting to a computer. Heathkit stopped selling kits a long time ago, but they continued to sell the factory−built version of the clock until 1995, when it was also discontinued. For Heathkit nostalgia (not including the clock) see http://www.heathkit−museum.com. The Heathkit company still exists, selling educational materials. See http://www.heathkit.com.

According to Dave Mills, Heathkit's patent on the "Most Accurate Clock" is due to expire soon, so maybe someone out there would like to clone it as a single−chip IC.

The NTP website has a DSP program (and a PDF file describing it) at http://www.eecis.udel.edu/~mills/resource.htm that decodes the WWV time signal using a shortwave radio and the TAPR/AMSAT DSP−93, a DSP kit which is no longer available. It was based on the Texas Instruments TMS320C25 DSP chip. The TAPR website at http://www.tapr.org includes a lot of information on homebrew DSP programming.

`Ntpd` includes a driver (type 6) for the IRIG−B and IRIG−E time codes, using `/dev/audio` on a Sun SPARCstation, with a note that it is "likely portable to other systems". WWV uses the IRIG−H time code.

WWV is run by NIST, which has a website at http://www.boulder.nist.gov/timefreq/index.html. This site includes the text of "Special Publication 432", which describes their time and frequency services, at http://www.boulder.nist.gov/timefreq/pubs/sp432/sp432.htm. WWV broadcasts on 2.5, 5, 10, 15, and 20

Mhz.

4.3 GPS and the "Totally Accurate Clock"

GPS signals include the correct time, and some GPS receivers have serial ports. `Ntpd` includes drivers for several GPS receivers. The 1PPS feature ("One Pulse Per Second", required for high accuracy) usually requires a separate interface to connect it to the computer.

TAPR (Tuscon Amateur Packet Radio) makes a kit for an interface called "TAC−2" (for "Totally Accurate Clock") that plugs into a serial port and works with any GPS receiver that can provide a 1PPS output−− including some "bare board" models that can be mounted directly to the circuit board. For more information see their website at http://www.tapr.org. The price (as of June 1999) is around $140, not including the GPS receiver. The kit does not include any enclosure or mounting hardware.

The CHU "gadget box" (described in another section) can also be used as an interface for the 1PPS signal. The NTP website has a discussion of this at http://www.eecis.udel.edu/~ntp/ntp_spool/html/pps.htm.

4.4 Low−frequency Time Signals: DCF77, MSF(Rugby), WWVB

These low−frequency stations broadcast a time code by simply switching the carrier on and off. Each station uses its own coding scheme, and summaries are available on the NTP website at http://www.eecis.udel.edu/~mills/ntp/index.htm (near the bottom of the page). DCF77 in Germany broadcasts on 77.5kHz. MSF in England (also called "Rugby", which apparently refers to its location) and WWVB in Colorado both broadcast on 60 kHz.

Reception of WWVB varies, but there are plans to increase its broadcast power, in several stages. You can follow its progress on NIST's website at http://www.boulder.nist.gov/timefreq/wwvstatus.html.

Inexpensive receivers that can plug into a serial port are reported to be available in Europe. `Ntpd` includes drivers for a couple of MSF receivers.

A number of companies in the US sell relatively inexpensive clocks that have built−in WWVB receivers (including several analog wall clocks), but I'm only aware of two that can be connected to a computer:

The Ultralink Model 320 sells for about $120 (as of June 1999) and has a serial interface and a straightforward ASCII command set, so it shouldn't be too hard to program. It draws 1mA from the serial port for power. The antenna can be up to 100 feet away from the computer, and the unit contains its own clock to maintain the time if it loses the signal. They also sell a "bare board" version for about $80 that is designed to work with the "BASIC Stamp" series of microcontrollers. See http://www.ulio.com/timepr.html.

Arcron Technology sells a desk clock with an optional serial port for about $130, including software for Windows. See