The Clock Mini−HOWTO

Table of Contents

The Clock Mini−HOWTO

Ron Bean, `rbean@execpc.com`

v2, July 1999

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

1.Introduction

The Real−Time−Clock (RTC) chips used on PC motherboards (and even expensive workstations) are notoriously inaccurate, usually gaining or losing a consistent 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 "how to set up linux" documentation, and it can't be set up automatically without 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 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 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, but I've included detailed instructions for the old `clock(8)` program at the end for anyone still running an older system.

Note

> You must be logged in as "**root**" to run any program that affects the RTC or the system time. Keep this in mind when setting up any of the programs described here.

Note

> If you run more than one OS on your machine, you should only let one of them set the CMOS clock, so they don't confuse each other. This includes the twice−a−year adjustment for Daylight Savings Time.

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.

2.Basic Timekeeping under Linux

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. 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 the 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.

It might seem obvious that in this case 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 have 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.

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.

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 radio clocks and such, 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 to have them automatically sync their clocks to each other.

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 places). Some distributions also include a graphical clock setting program that runs in an X−window, but they're 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 clock (using another program to set the time 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`.

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".

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.

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.

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.freenet.co.uk/chrony/index.html. He distributes the program 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.

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−− all you need is 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.

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 http://www.arctime.com

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.

5.Detailed instructions for clock(8)

This section is retained from the earlier version of this mini−HOWTO, and is provided for anyone still using the old `clock(8)` program. Everything you need to know is in the man page, but the following discussion will walk you through the process.

> ***Note***
>
> You must be **root** to run "`clock`", or any other program that affects either the system time or the CMOS clock.

5.1 Checking your installation

Check your system startup files for a command like "`clock −a`" or "`clock −ua`". Depending on which distribution you're using, it might be in `/etc/rc.local`, or `/etc/rc.d/rc.sysinit`, or some similar place.

If it says "`clock −s`" or "`clock −us`", change the "`s`" to an "`a`", and then check to see if you have the file `/etc/adjtime`, which contains a single line that looks something like this:

```
0.000000 842214901 0.000000
```

These numbers are the correction factor (in seconds per day), the time the clock was last corrected (in seconds since Jan 1, 1970), and the partial second that was rounded off last time. If you don't have this file, login as **root** and create it, with a single line that looks like this (all zeros):

```
0.0 0 0.0
```

Then run "`clock -a`" or "`clock -ua`" manually from the shell to update the 2nd number (use the "`u`" if your clock is set to Universal instead of local time).

5.2 Measuring your clock's drift rate

First, you need to know what time it is `:-)`. Your local time of day number may or may not be accurate. My favorite method is to call WWV's voice announcment at (303)499−7111 (this is a toll call). If you have access to a network time server, you can use the `ntpdate` program from the `xntpd` package (use the `-b` flag to keep the kernel from messing with the CMOS clock). Otherwise use "`date -s hh:mm:ss`" to set the kernel time by hand, and then "`clock -w`" to set the CMOS clock from the kernel clock. You'll need to remember when you last set the clock, so write down the date someplace where you won't lose it. If you used `ntpdate`, do "`date +%s`" and write down the number of seconds since Jan 1,1970.

Then come back some days or weeks later and see how far the clock has drifted. If you're setting the clock by hand, I'd recommend waiting at least two weeks, and only calculate the drift rate to the nearest .1 sec/day. After several months you could get to the nearest .01 sec/day (some people claim more accuracy than that, but I'm being conservative here). If you use `ntpdate` you shouldn't have to wait that long, but in any case you can always fine−tune it later.

You can have cron run "`clock -a`" at regular intervals to keep the system time in line with the (corrected) CMOS time. This command will also be run from your startup file every time you boot the system, so if you do that often (as some of us do), that may be enough for your purposes.

Note that certain programs may complain if the system time jumps by more than one second at a time, or if it jumps backwards. If you have this problem, you can use `xntpd` or `ntpdate` to correct the time more gradually.

5.3 Example

To set time

Login as **root**. Dial (303)499−7111 (voice), listen for time announcement. Then type:

```
date -s hh:mm:ss
```

but don't press enter until you hear the beep. (You could use "`ntpdate`" here instead of "`date`", and skip the phone call) This sets the "kernel time". Then type:

```
clock -w
```

This sets the CMOS time to match the kernel time. Then type:

```
date +%j
```

(or "`date +%s`" if you used "`ntpdate`" instead of "`date`" above) and write down the number it gives you for next time.

To reset time and check drift rate

Find the date you wrote down last time. Login as **root** Then type:

```
clock −a
```

This sets the kernel time to match the current CMOS time. Dial (303)499−7111 (voice), listen for announcement. Then type:

```
date
```

and press enter when you hear the beep, but while you're waiting, write down the time they announce, and don't hang up yet. This tells you what time your machine thought it was, when it should have been exactly on the minute. Now type in

```
date hh:mm:00
```

using the minute ***after*** the one that was just announced, and press enter when you hear the beep again (now you can hang up). For `hh` use the local hour. This sets the "kernel time". Then type:

```
clock −w
```

which writes the new (correct) time to the CMOS clock. Now type:

```
date +%j
```

(or "`date +%s`" if that's what you used before)

You now have three numbers (two dates and a time) that will allow you to calculate the drift rate.

Calculating the correction factor

When you ran "`date`" on the minute, was your machine slow or fast? If it was fast, you'll have to subtract some number of seconds, so write it down as a negative number. If it was slow, you have to add some seconds, so write it down as positive.

Now subtract the two dates. If you used "`date +%j`", the numbers represent the day−of−year (1−365, or 1−366 in leap years). If you've passed Jan 1 since you last set the clock you'll have to add 365 (or 366) to the 2nd number. If you used "`date +%s`" then your number is in seconds, and you'll have to divide it by 86400 to get days.

If you already had a correction factor in `/etc/adjtime`, you'll have to account for the number of seconds you've already corrected. If you've overcorrected, this number will have the opposite sign of the one you just measured; if you've undercorrected it will have the same sign. Multiply the old correction factor by the number of days, and then add the new number of seconds (signed addition−− if the two numbers have the same sign, you'll get a larger number, if they have opposite signs, you'll get a smaller number).

Then divide the total number of seconds by the number of days to get the new correction factor, and put it in `/etc/adjtime` in place of the old one. Write down the new date (in seconds or days) for next time.

Here's what my `/etc/adjtime` looks like:

```
-9.600000 845082716 -0.250655
```

(note 9.6 seconds per day is nearly five minutes per month!)
