Calendar in the command line

DATE COMMANDER

Anka Draganski

The legacy cal and date tools help users keep track of the time and date. You can even change the system time with a single shell command.

BY HEIKE JURZIK

Wall calendars are pretty, and PDAs are handy, but there is nothing like Linux to help you track the time. Linux gives a quick overview of the date and time with the *cal* and *date* command line tools.

You can use *cal* to draw a calendar sheet or an overview for a full year in a terminal window, and *date* not only outputs the date and time in various formats, but can also adjust the system time, assuming you have administrative privileges.

Command Line

Although GUIs such as KDE or GNOME are useful for various tasks, if you intend to get the most out of your Linux machine, you will need to revert to the good old command line from time to time. Apart from that, you will probably be confronted with various scenarios where some working knowledge will be extremely useful in finding your way through the command line jungle.

A cal for all Seasons

The *cal* tool outputs a calendar in the command line. Entering *cal* without any additional parameters gives you a neatly-formated overview of the current month. If you want to output an overview for the whole year, simply add the *-y* option. And you can set the *-3* option (Figure 1) to display the current month, along with the previous and following months.

cal shows you a specific month if you add a one or two-digit number (such as *6* or *06*) and a four-digit year (e.g. 2005) to the command. For example, to get *cal* to show you June 2005, you would enter:

```
cal 06 2005
```

As *cal* goes back to January 1, 0001, you can't get away with typing a two-digit year. For example,

```
cal 06 05
```

would give you June in the 5 year AD.

By default *cal* starts the week with a Sunday on some Linux versions – if you prefer to start the week on a Monday, just add *-m* to the *cal*-command.

The *-j* parameter tells *cal* to count the days from 1 365 (or 366 in leap years):

```
$ cal -j
        June 2005
 Su  Mo  Tu  We  Th  Fr  Sa
            152 153 154 155
156 157 158 159 160 161 162
163 164 165 166 167 168 169
...
```

This tells you that June 19 is the 170th day in the year, for example.

Time Pundits

The *date* program not only has a calendar function, it tells you more about the time. Typing the command at the Shell prompt without specifying any options gives you the current date and time:

```
$ date
Sun Jun 19 18:38:15 EDT 2005
```

If the output is in a language you don't understand, you can use the *echo* command to check the value of the *LC_TIME* **environmental variable**, and set the variable to an appropriate value (typing *locale -a* shows you the locale-specific settings available for your system).

```
$ echo $LC_TIME
$ export LC_TIME=en_US
$ date
Sun Jun 19 19:00:15 EDT 2005
```

You can use *date* with the environmental variable *TZ* (for "time zone") to display the current date and time in a different country. The */usr/share/zoneinfo* directory lists all the timezones in the world (some of them in subdirectories.) If you want to know what time it is in New Zealand, for example, you could temporarily set the *TZ* variable, just for one call to *date*:

```
$ TZ=NZ date
Mon Jun 20 11:00:44↵
NZST 2005
```

If you have friends or relatives abroad, and you want to keep from waking them up in the middle of the night, you can avoid all that typing by defining an alias for the complete command. To define an alias, add something like the following command to your *~/.bashrc*:

```
alias nz='TZ=NZ date'
```

Then reparse the configuration file as follows,

```
source ~/.bashrc
```

or just log off and back on again. The next time you need to know what time it is in New Zealand, you can just type *nz* to find out.

Talking Clock

The *date* program has a few useful output formatting parameters. Preceded by a plus sign (+) various combinations of percent and letters will give you the format you need:

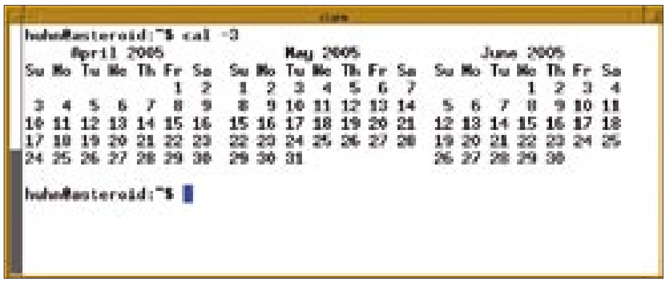
```
$ date '+Today is ↵
day %d in the month ↵
of %B of the year %Y.'

Today is day 19 in↵
the month of June ↵
of the year 2005.
```

%d gives you a two-digit format for the day, *%B* writes out the name of the month in full, and *%Y* gives you the year (as a four-digit number.) Table 1 shows an overview of the most important formatting wildcards.

Table 1: Output Format for the date command

%H	Hours (*00* through *23*)
%I	Hours (*01* through *12*)
%M	Minutes (*00* through *59*)
%S	Seconds (*00* through *60*); this range actually has 60 rather than 59 seconds due to the leap second.
%p	am or pm (*AM, PM*)
%r	Twelve hour clock (*hh:mm:ss AM/PM*)
%R	24 hour clock (*hh:mm:ss*), *%H:%M*
%s	Seconds after January 1, 1970 (Unix epoch)
%Z	Current timezone (*EDT, GMT* etc.)
%a	Weekday short form (*Sun, Mon* etc.)
%A	Weekday long form (*Sunday, Monday* etc.)
%b	Month short form (*Jan, Feb* etc.)
%B	Month long form (*January, February* etc.)
%d	Two-digit day (*01, 31* etc.)
%e	Day (single digit with blank, e.g. *7*)
%D	Date format *mm/dd/yy*
%j	Shows the day of the year – that is *1* for January 1, and (except in leap years) *365* for December 31.
%u	Shows the day of the week (*1* through *7*).
%U	Shows the calendar week (*00* through *53*).
%y	Year, two-digit (*00* through *99*)
%Y	Year, four-digit (e.g. *1973*)
%%	Prints a percent sign.
%n	End-of-line
%t	Tabulator



```
huhn@asteroid:~$ cal -3
     April 2005             May 2005              June 2005
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                1  2   1  2  3  4  5  6  7            1  2  3  4
 3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
24 25 26 27 28 29 30  29 30 31              26 27 28 29 30

huhn@asteroid:~$
```

Figure 1: cal -3 displays the previous, current, and following months.

It always makes sense to use these options when you need part of the current date and time. For example, you could use these parameters in combination with *tar* and *date* to create backups automatically named for the day, month, and year they were created:

```
$ tar cvfj ➥
backup_-_$(date +'%d')_$➥
(date +'%m')_$(date +'%y')➥
.tar.bz2 *
```

Tomorrow Never Comes...

You can use the *-d* option to specify the *date* output. The parameter introduces a whole bunch of keywords and numbers, allowing you to use keywords such as *tomorrow* and *yesterday*, for example:

```
$ date -d tomorrow
Mon Jun 20 18:37:10 EDT 2005
$ date -d yesterday
Sat Jun 18 18:37:55 EDT 2005
```

There are more keywords that support relative references (*next*, *ago*, etc.), fixed dates (*09 June 1973*, *09 Jun 73* etc.), weekdays (*Monday*, *Tuesday*, etc.) and periods of time (*day*, *year*, *week*, *month*, *fortnight*, etc.), so if you really wanted to, you could output yesterday's date by entering

```
date -d '1 day ago'
```

If you use multiple keywords, like in our example, you must use either single ('') or double quotes (""). In combination with the output formating options you could output last month

```
$ date -d '1 month ago' ➥
+'Last month: %B'
Last month: May
```

or find out what weekday January 3 was:

```
$ date -d ➥
'third➥
January'➥
 +%A

Monday
```

Those of you who need to keep to deadlines can work out how many days you have left to complete your current project with a single command. This means leveraging Bash's legendary arithmetic skills; the expression we want to calculate is in two sets of parentheses, e.g.,

```
$ echo $((1-1))
0
```

To discover the number of days to your deadline, you need two *date* expressions in the parentheses: the date of the deadline and today's date, which will be subtracted:

```
$ echo $((`date -d '29 ➥
September 2005' +%j`➥
-`date +%j`))
102
```

The + *%j* formating option tells *date* to count the days; I'll leave it up to you to work out today's date!

This syntax only works if both days are in the same year, but there is a workaround. Just replace the day counter (*%j*) with the Unix time (*%s*), which counts the number of seconds since 01.01.1970 (0:00 hours). Then divide the difference in seconds by 60*60*24 = 86400 to convert the result to days:

```
$ echo "$((`date -d ➥
'19 June 2006' ➥
+%s`-`date +%s`))➥
 / 86400" | bc -l
364.21625000000000000000
```

Setting the Time

As an administrator, you can use *date* to give you a convenient approach to setting the system time. You can simply type the date and time as a single large number following the *date* command (without a plus sign.) The rules for writing the date and time are quite strict: first, you need the month (two digits), then the day (two digits), the hour (two digits), and optionally the first two digits of the year (e.g., *19*), the last two digits of the year (e.g. *73*), and the seconds (two digits). To set the time to June 9, 1973, 08:13, you would first enter *su -* to become *root*, and then type the following:

```
# date 060908131973
Sat Jun 09 08:13:00 EDT 1973
```

As an alternative, you could set the *-s* parameter and use the keywords and numbers described in the previous section titled *Tomorrow Never Comes…*. For example, the following command would put the clock two minutes forward:

```
date -s '+2 minutes'
```

It is just as easy to put the calendar back one day:

```
date -s 'yesterday'
```

Note that changing the time does not affect your CMOS clock. PCs have a battery-powered clock that keeps the time and date even when the computer isn't running. As this clock is located on the same chip as the CMOS RAM, it is referred to as the CMOS clock. The CMOS clock is also called the hardware clock, RTC (Real Time Clock), or BIOS clock. To change the hardware clock to the current system time, again working as *root*, enter the following command:

```
hwclock -w
```

Some programs do not take kindly to big time jumps, for example, the *xchat* IRC tool. If you want to experiment with *date*, you might discover that you need to restart some programs. ■

THE AUTHOR

Heike Jurzik studied German, Computer Science and English at the University of Cologne, Germany. She discovered Linux in 1996 and has been fascinated with the scope of the Linux command line ever since. In her leisure time you might find Heike hanging out at Irish folk sessions or visiting Ireland.