Examples of `date`

Here are a few examples. Also see the documentation for the `-d`option in the previous section.

- To print the date of the day before yesterday:

```
date --date='2 days ago'
```

- To print the date of the day three months and one day hence:

```
date --date='3 months 1 day'
```

- To print the day of year of Christmas in the current year:

```
date --date='25 Dec' +%j
```

- To print the current full month name and the day of the month:

```
date '+%B %d'
```

  But this may not be what you want because for the first nine days of the month, the '`%d`' expands to a zero-padded two-digit field, for example '`date -d 1may '+%B %d''` will print '`May 01`'.

- To print a date without the leading zero for one-digit days of the month, you can use the (GNU extension) '`-`' flag to suppress the padding altogether:

```
date -d 1may '+%B %-d'
```

- To print the current date and time in the format required by many non-GNU versions of `date` when setting the system clock:

```
date +%m%d%H%M%Y.%S
```

- To set the system clock forward by two minutes:

```
date --set='+2 minutes'
```

- To print the date in Internet RFC 5322 format, use ‘`date --rfc-email`’. Here is some example output:

```
Fri, 09 Sep 2005 13:51:39 -0700
```

- To convert a date string to the number of seconds since the epoch (which is 1970-01-01 00:00:00 UTC), use the `--date`option with the ‘`%s`’ format. That can be useful in sorting and/or graphing and/or comparing data by date. The following command outputs the number of the seconds since the epoch for the time two minutes after the epoch:

```
date --date='1970-01-01 00:02:00 +0000' +%
120
```

If you do not specify time zone information in the date string,`date` uses your computer's idea of the time zone when interpreting the string. For example, if your computer's time zone is that of Cambridge, Massachusetts, which was then 5 hours (i.e., 18,000 seconds) behind UTC:

```
# local time zone used
date --date='1970-01-01 00:02:00' +%s
18120
```

- If you're sorting or graphing dated data, your raw date values may be represented as seconds since the epoch. But few people can look at the date '`946684800`' and casually note "Oh, that's the first second of the year 2000 in Greenwich, England."

```
date --date='2000-01-01 UTC' +%s
946684800
```

An alternative is to use the `--utc` (`-u`) option. Then you may omit '`UTC`' from the date string. Although this produces the same result for '`%s`' and many other format sequences, with a time zone offset different from zero, it would give a different result for zone-dependent formats like '`%z`'.

```
date -u --date=2000-01-01 +%s
946684800
```

To convert such an unwieldy number of seconds back to a more readable form, use a command like this:

```
# local time zone used
date -d '1970-01-01 UTC 946684800 seconds'
1999-12-31 19:00:00 -0500
```

Or if you do not mind depending on the '@' feature present since coreutils 5.3.0, you could shorten this to:

```
date -d @946684800 +"%F %T %z"
1999-12-31 19:00:00 -0500
```

Often it is better to output UTC-relative date and time:

```
date -u -d '1970-01-01 946684800 seconds'
2000-01-01 00:00:00 +0000
```

- Typically the seconds count omits leap seconds, but some systems are exceptions. Because leap seconds are not predictable, the mapping between the seconds count and a future timestamp is not reliable on the atypical systems that include leap seconds in their counts.

  Here is how the two kinds of systems handle the leap second at 2012-06-30 23:59:60 UTC:

```
# Typical systems ignore leap seconds:
date --date='2012-06-30 23:59:59 +0000' +%
1341100799
date --date='2012-06-30 23:59:60 +0000' +%
date: invalid date '2012-06-30 23:59:60 +0
date --date='2012-07-01 00:00:00 +0000' +%
1341100800
```

```
# Atypical systems count leap seconds:
date --date='2012-06-30 23:59:59 +0000' +%
1341100823
date --date='2012-06-30 23:59:60 +0000' +%
1341100824
date --date='2012-07-01 00:00:00 +0000' +%
1341100825
```