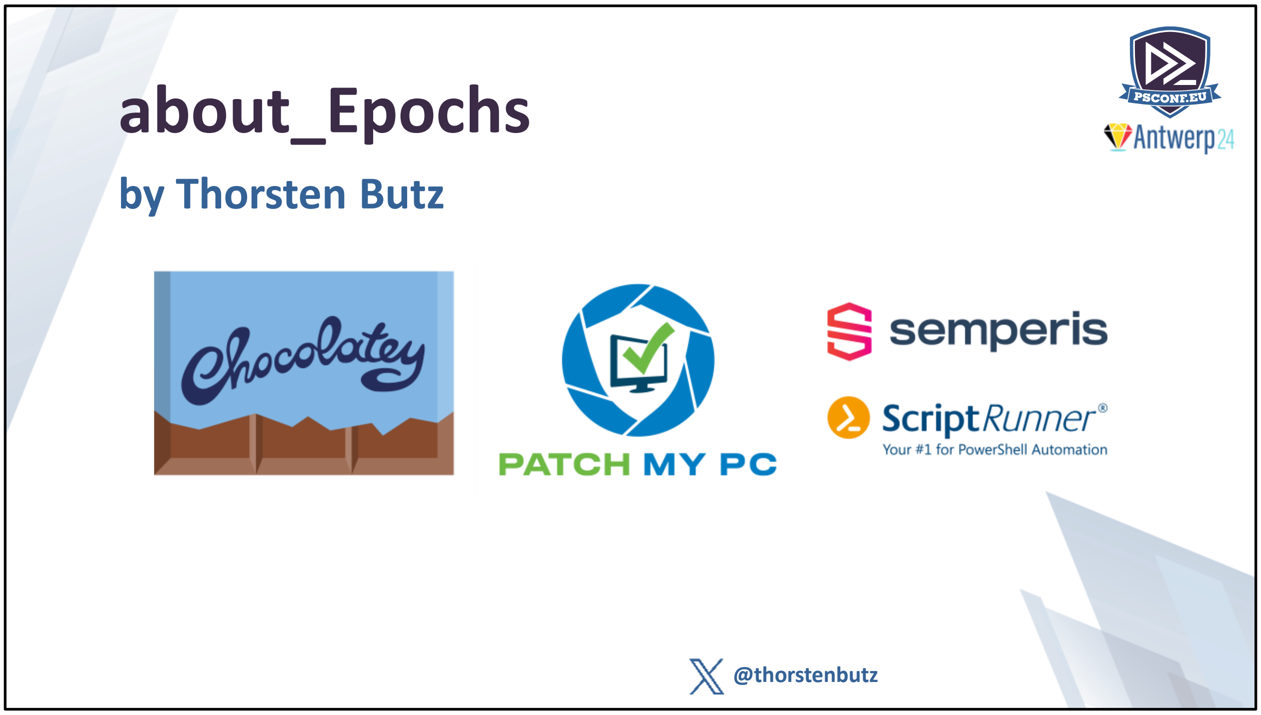


about_Epochs

by Thorsten Butz

PSCONF.EU
Antwerp24
Chocolatey
PATCH MY PC
semperis
ScriptRunner®
Your #1 for PowerShell Automation
@thorstenbutz

about Epochs (a PoShaKucha)
What do you think when you read the numbers 1, 1601 or 1970?

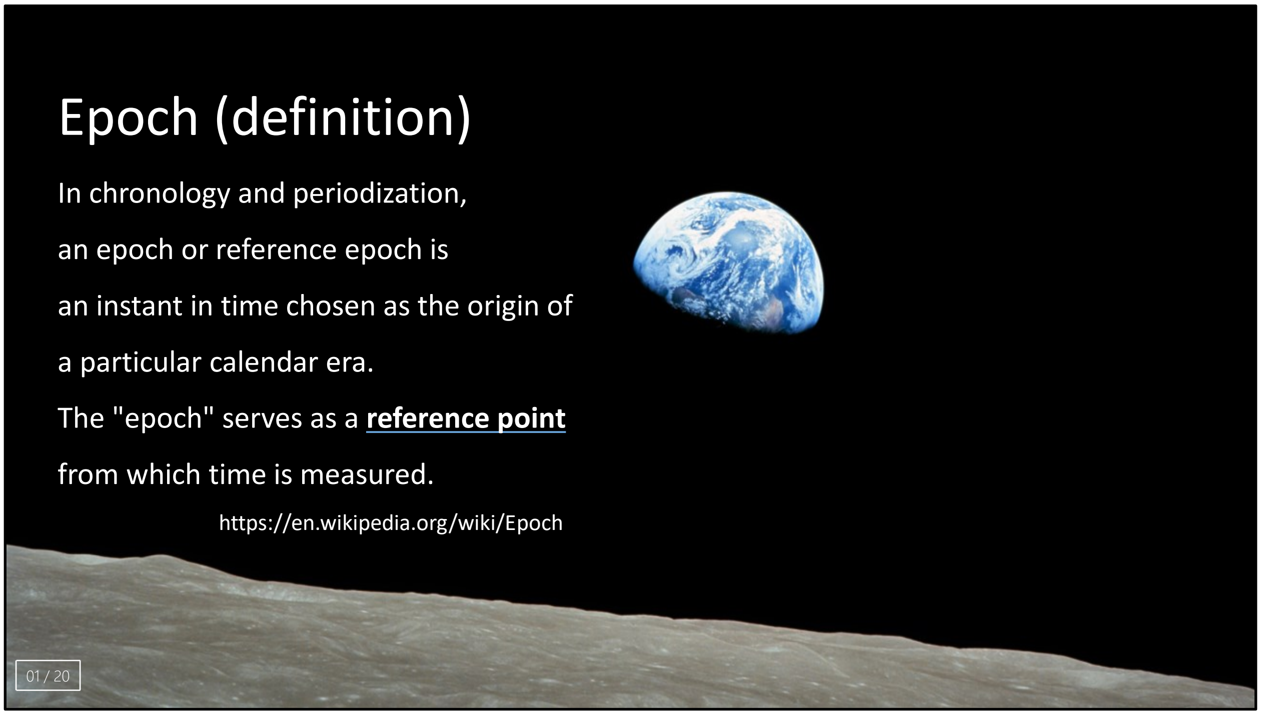
Have you ever thougth it would simplify your life if 2 seconds made a minute, 2 minutes made an hour and 2 hours made a day?

Time formats are one of the trickiest and most error-prone things in the life of a computer scientist. So it's no wonder that PowerShell and the operating systems on which it runs are full of turbulences and steep drop-offs.

Fancy some nerdy PowerShell knowledge at warp speed? After you watched this, you will then get to know a few very special bon mots about time representation in your favorite shell and also learned practical things like how to read the infamous AD timestamps correctly.

about PechaKucha/PoShaKucha
PechaKucha is a storytelling format in which a presenter shows 20 slides for 20 seconds of commentary each. PoShaKucha is a play on words derived from the term "PoSh" (for PowerShell) and the PechaKucha.

Read more about it
https://psconf.eu/introducing-poshakucha-a-fusion-of-powershell-and-pechakucha/
https://github.com/thorstenbutz/PoShaKucha



Epoch (definition)

In chronology and periodization,

an epoch or reference epoch is

an instant in time chosen as the origin of

a particular calendar era.

The "epoch" serves as a **reference point**

from which time is measured.

https://en.wikipedia.org/wiki/Epoch

01 / 20

Epcoch (computing)
"In computing, an **epoch** is a fixed date and time used as a reference from which a computer measures system time. Most computer systems determine time as a number representing the seconds removed from a particular arbitrary date and time.

For instance, Unix and POSIX measure time as the number of seconds that have passed since Thursday 1 January 1970 00:00:00 UT, a point in time known as the Unix epoch. Windows NT systems, up to and including Windows 11 and Windows Server 2022, measure time as the number of 100-nanosecond intervals that have passed since 1 January 1601 00:00:00 UTC, making that point in time the epoch for those systems.[1] Computing epochs are almost always specified as midnight Universal Time on some particular date."
https://en.wikipedia.org/wiki/Epoch_(computing)

Background picture
"Earthrise was taken by astronaut William Anders during the Apollo 8 mission, the first crewed voyage to orbit the Moon. Before Anders found a suitable 70 mm color film, mission commander Frank Borman took a black-and-white photograph of the scene, with the Earth's terminator touching the horizon. The land mass position and cloud patterns in this image are the same as those of the color photograph entitled Earthrise. .."

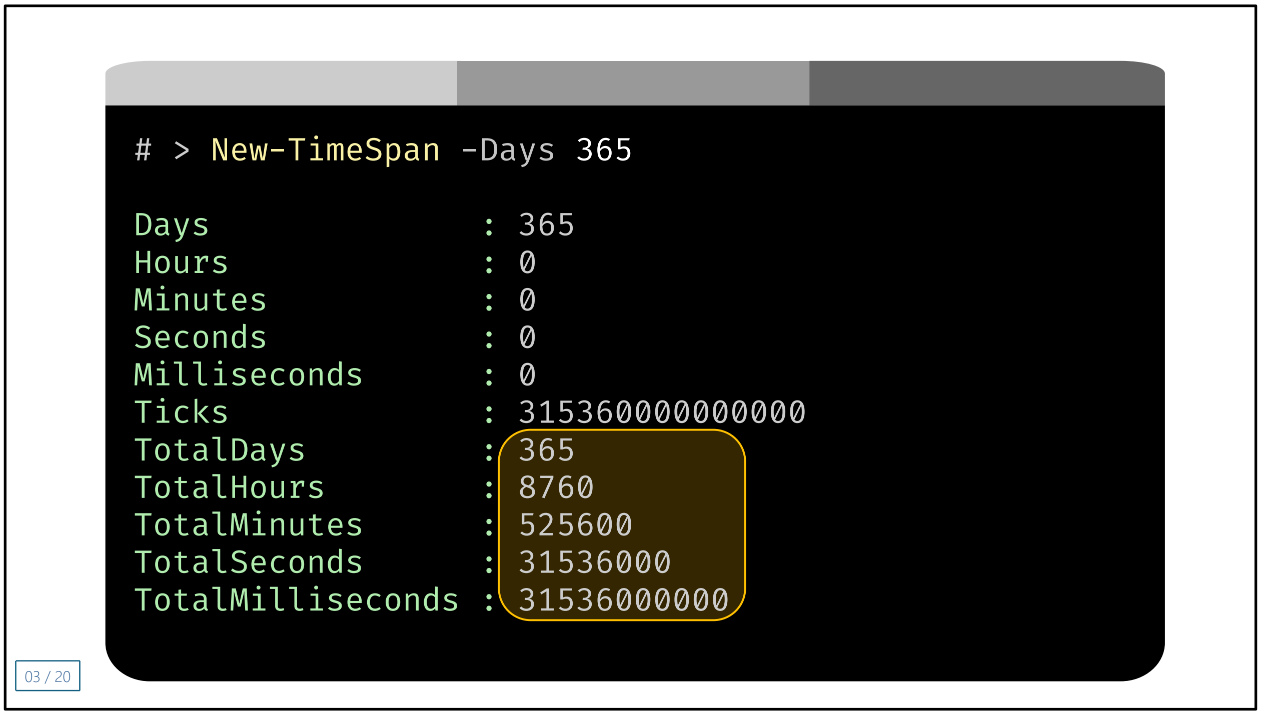
The photograph was taken from lunar orbit on December 24, 1968, 16:39:39.3 UTC,
https://en.wikipedia.org/wiki/Earthrise

Beat
In music the beat is the basic unit of time, the pulse (regularly repeating event).
The beat can also be seen as a metaphor for time, measuring intervals that you might call a seconds.

By the way
Did you know how 1 second is formally defined?

This is the current and formal definition in the International System of Units (SI), taken form the Wikipedia:
"The second is defined by taking the fixed numerical value of the caesium frequency,
ΔνCs, the unperturbed ground-state hyperfine transition frequency of the caesium 133 atom, to be 9192631770 when expressed in the unit Hz, which is equal to s−1."

https://en.wikipedia.org/w/index.php?title=Second



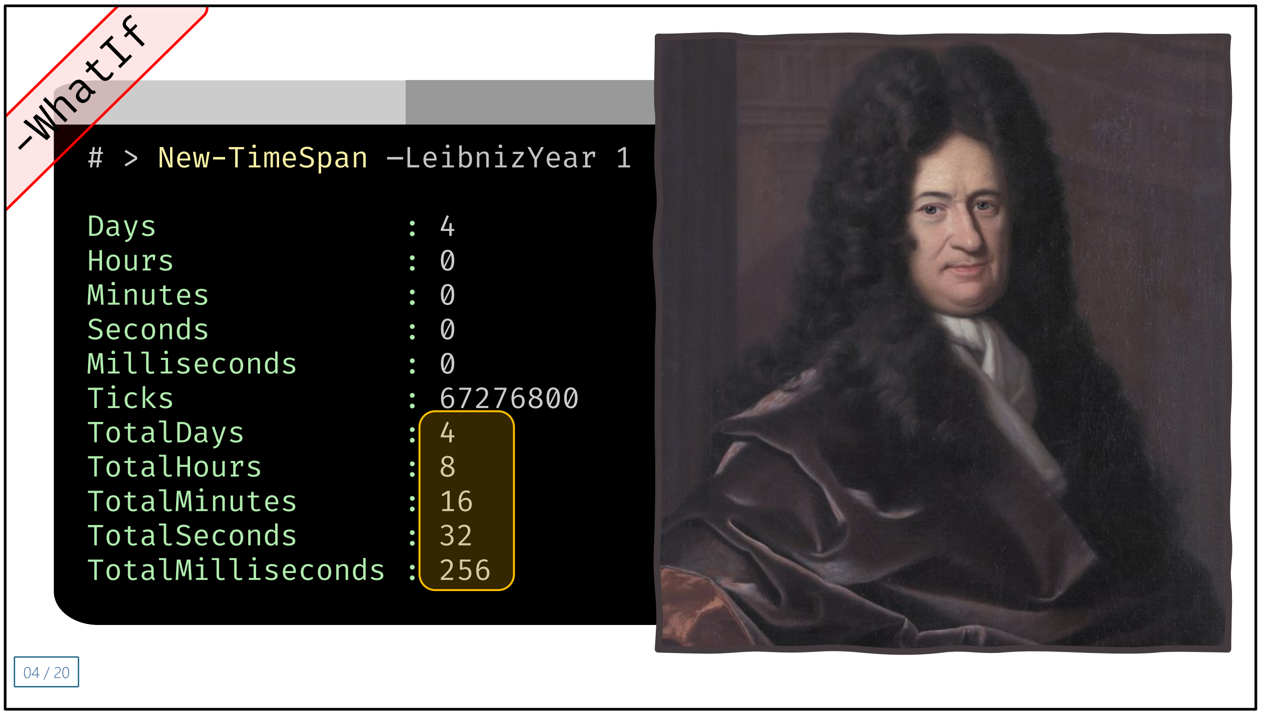
```
# > New-TimeSpan -Days 365

Days              : 365
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 0
Ticks             : 315360000000000
TotalDays         : 365
TotalHours        : 8760
TotalMinutes      : 525600
TotalSeconds      : 31536000
TotalMilliseconds : 31536000000
```

03 / 20

A year on Earth is the time taken for Earth to revolve around the sun.
A bit more general: A year is the time taken for astronomical objects to complete one orbit.

People began very early on to derive calendars from observing the sky and its recurring cycles.

"Archeologists have reconstructed methods of timekeeping that go back to prehistoric times at least as old as the Neolithic. The natural units for timekeeping used by most historical societies are the day, the solar year and the lunation. Calendars are explicit schemes used for timekeeping. The first historically attested and formulized calendars date to the Bronze Age, dependent on the development of writing in the ancient Near East. In 2000 AD, Victoria, Australia, a Wurdi Youang stone arrangement could date back more than 11,000 years.
In 2013, archaeologists unearthed ancient evidence of a 10,000-year-old calendar system in Warren Field, Aberdeenshire. This calendar is the next earliest, or "the first Scottish calendar". The Sumerian calendar was the next earliest, followed by the Egyptian, Assyrian and Elamite calendars."
https://en.wikipedia.org/wiki/History_of_calendars



```
# > New-TimeSpan -LeibnizYear 1

Days              : 4
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 0
Ticks             : 67276800
TotalDays         : 4
TotalHours        : 8
TotalMinutes      : 16
TotalSeconds      : 32
TotalMilliseconds : 256
```

What if a computer scientist had invented / designed time?
Gottfried Wilhelm Leibniz, the "godfather" of PowerShell, made the binary numbering system popular. What if ... (a mind game)

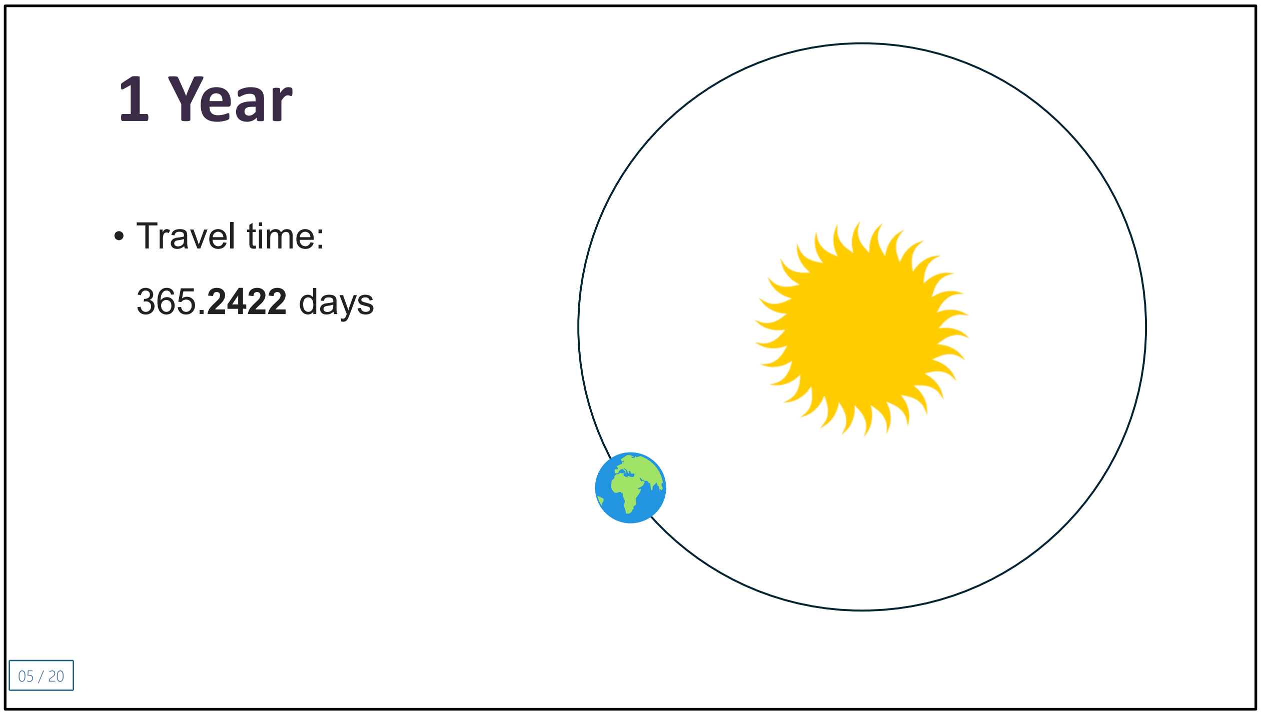
What if
- 1 year equals 2 months
- 1 month eqlas 2 days
- 1 day equals 2 hous
- 1 hour equals 2 minutes
- 1 minute equals 2 seconds
- 1 seconds equals 2^3 milliseconds

You see a series of familiar **powers of two** highlighted.

Calculation:
a) Our current system (Days, Hours, Minutes, Seconds, Milliseconds)
`365 * 24 * 60 * 60 * [Math]::Pow(10,3)`

b) The Leibniz year, base 2 (Days, Hours, Minutes, Seconds, Milliseconds)
`4 * 2 * 2 * 2 * [Math]::Pow(2,3)`

Picture of Gottfried Wilhelm Leibniz
by Christoph Bernhard Francke - Herzog Anton Ulrich-Museum Braunschweig, Public Domain, https://commons.wikimedia.org/w/index.php?curid=57268659



1 Year

- Travel time:

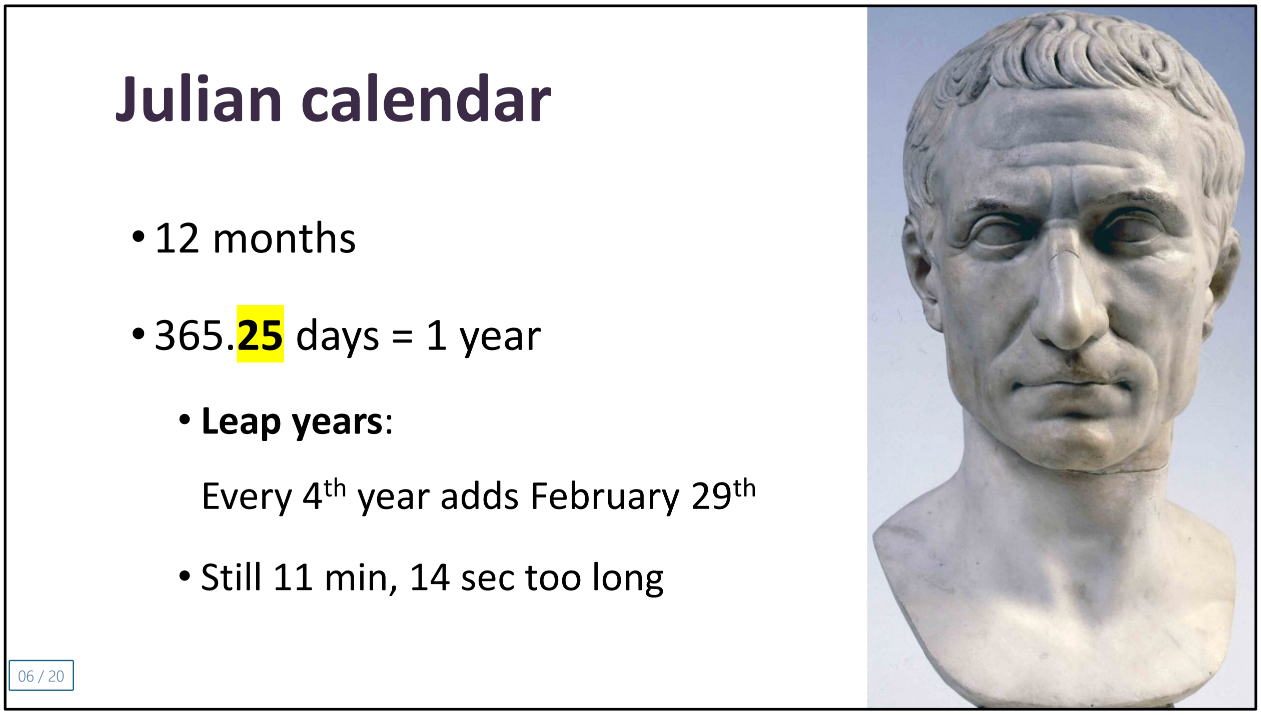
  365.**2422** days

05 / 20

What defines 1 day?
"A day is the time period of a full rotation of the Earth with respect to the Sun. On average, this is 24 hours (86,400 seconds). As a day passes at a given location it experiences morning, noon, afternoon, evening, and night. "
https://en.wikipedia.org/wiki/Day

What defines 1 year?
"A year is the time taken for astronomical objects to complete one orbit. For example, a year on Earth is the time taken for Earth to revolve around the Sun. [..]
Due to the Earth's axial tilt, the course of a year sees the passing of the seasons, marked by changes in weather, the hours of daylight, and, consequently, vegetation and soil fertility. In temperate and subpolar regions around the planet, four seasons are generally recognized: spring, summer, autumn, and winter. In tropical and subtropical regions, several geographical sectors do not present defined seasons; but in the seasonal tropics, the annual wet and dry seasons are recognized and tracked."
https://en.wikipedia.org/wiki/Year

Great explanations and demonstrations (in German)
https://kinder.wdr.de/tv/die-sendung-mit-der-maus/av/video-sonne-und-erde-100.html
https://www.wdrmaus.de/filme/sachgeschichten/29_Februar.php5



Julian calendar

- 12 months
- 365.**25** days = 1 year
  - **Leap years**:

    Every 4th year adds February 29th
  - Still 11 min, 14 sec too long

06 / 20

The Julian calendar, proposed by **Julius Caesar**, is a solar calendar of 365 days in every year with an additional leap day every fourth year. The Julian calendar is still used as a religious calendar in parts of the world.

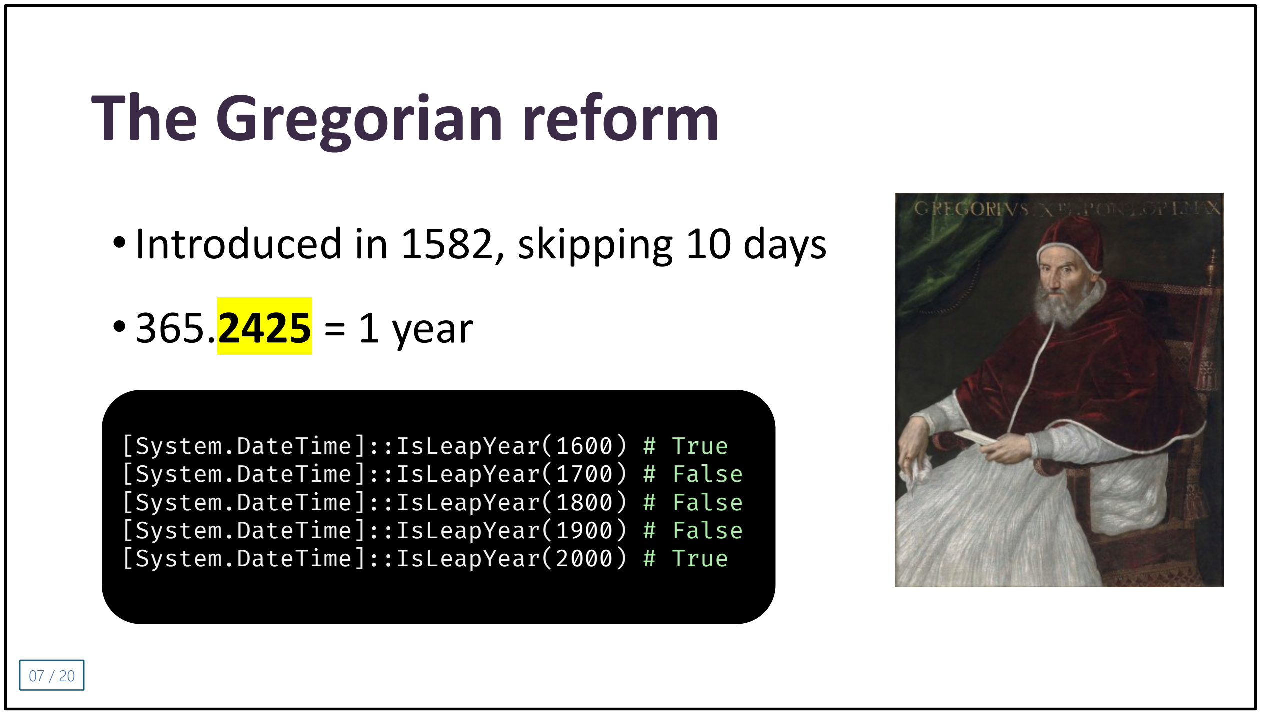
According to today's calendar, the Julian calendar was proposed in 46 BCE (before common era) and took effect on 1 January 45 BCE.

The calendar is based on its predecessor, the also roman Numa calendar.

Steve Morse, the inventor of the 8086 CPU, provides lots of details. Highly recommended:
https://stevemorse.org/juliancalendar/julian.htm

A PowerShell module for conversions to the Julian Calendar:
https://www.powershellgallery.com/packages/JulianCalendar/1.3

Picture
https://en.wikipedia.org/wiki/Julius_Caesar#/media/File:Gaius_Iulius_Caesar_(Vatican_Museum).jpg



The Gregorian reform

- Introduced in 1582, skipping 10 days
- 365.**2425** = 1 year

```
[System.DateTime]::IsLeapYear(1600) # True
[System.DateTime]::IsLeapYear(1700) # False
[System.DateTime]::IsLeapYear(1800) # False
[System.DateTime]::IsLeapYear(1900) # False
[System.DateTime]::IsLeapYear(2000) # True
```

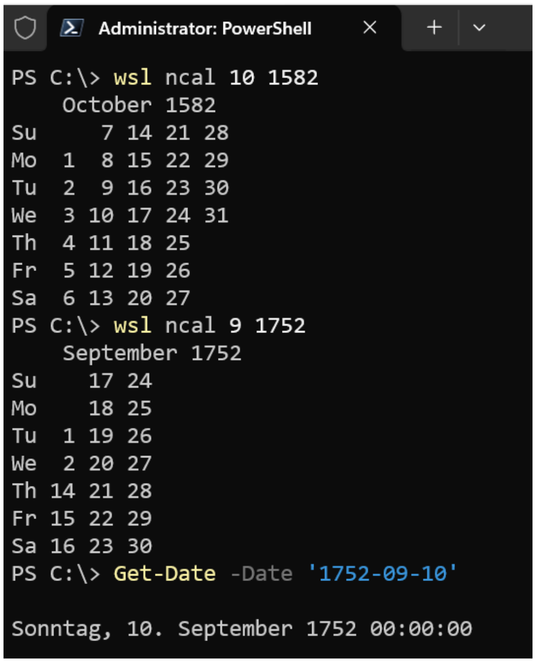
07 / 20

Since the Julian Calendar does not precisely represent the astronomical reality. A difference of 0.0078 days per year comes to 1 day every 128 years. That's about 3 days every 400 years." (Steve Morse: https://stevemorse.org/juliancalendar/julian.htm)

Fix 1
Pope Gregory XIII decreed that 10 days be stricken from the calendar, the cutover date for the calendar should be October 4, 1582.

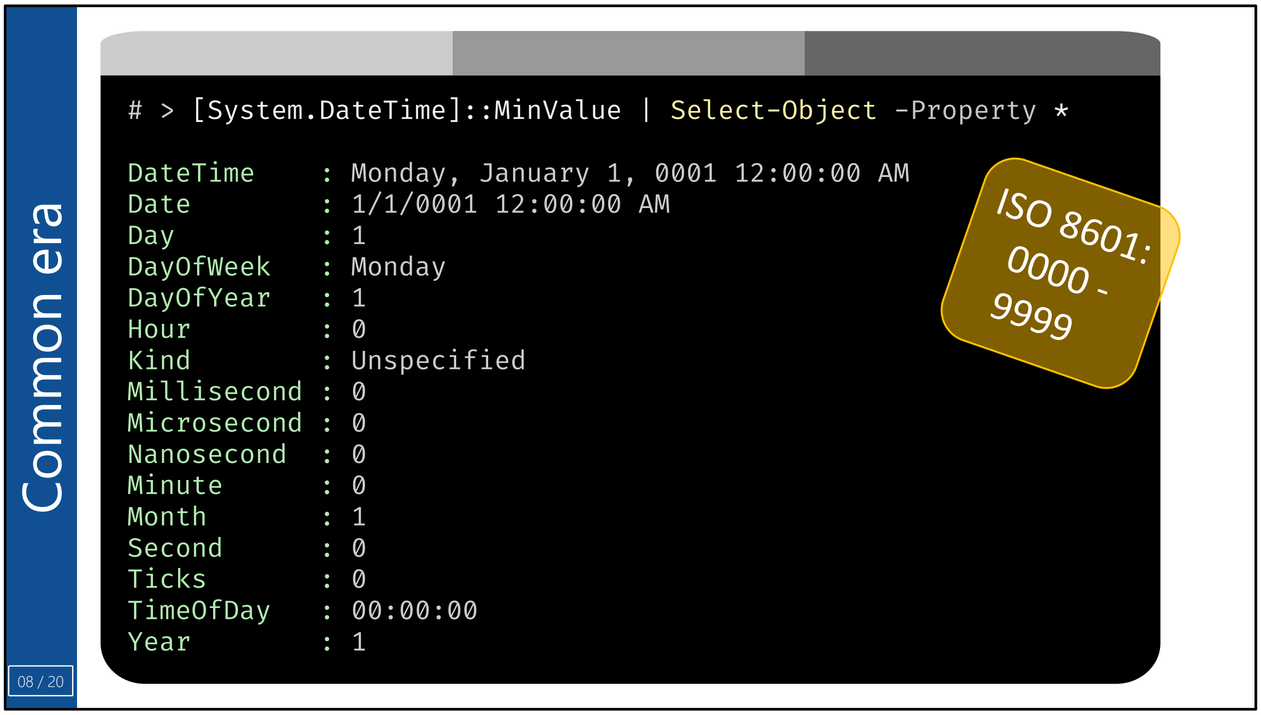
The screenshot displays "missing days" via the cal/ncal program in 1752. Different areas/countries opted in for the change over the following centuries, this reflects the adoption in the US.

Fix 2
The pope decreed that century years (those ending in 00) not be leap years unless they are evenly divisible by 400. That means that every 400 years we would have 3 fewer leap years.



```
PS C:\> wsl ncal 10 1582
    October 1582
Su     7 14 21 28
Mo  1  8 15 22 29
Tu  2  9 16 23 30
We  3 10 17 24 31
Th  4 11 18 25
Fr  5 12 19 26
Sa  6 13 20 27
PS C:\> wsl ncal 9 1752
    September 1752
Su    17 24
Mo    18 25
Tu  1 19 26
We  2 20 27
Th 14 21 28
Fr 15 22 29
Sa 16 23 30
PS C:\> Get-Date -Date '1752-09-10'

Sonntag, 10. September 1752 00:00:00
```

Picture of Pope Gregory
By Lavinia Fontana - Christie's, LotFinder: entry 6074459 (sale 13756, lot 563, London, 18 May 2017), Public Domain, https://commons.wikimedia.org/w/index.php?curid=77449113



Common era

```
# > [System.DateTime]::MinValue | Select-Object -Property *

DateTime    : Monday, January 1, 0001 12:00:00 AM
Date        : 1/1/0001 12:00:00 AM
Day         : 1
DayOfWeek   : Monday
DayOfYear   : 1
Hour        : 0
Kind        : Unspecified
Millisecond : 0
Microsecond : 0
Nanosecond  : 0
Minute      : 0
Month       : 1
Second      : 0
Ticks       : 0
TimeOfDay   : 00:00:00
Year        : 1
```

ISO 8601: 0000 - 9999

08 / 20

Our perception of time and time representation is basically an adoption of the Gregorian calender, taking the presumed nativity of Jesus Christ as the starting time.
While time measurement is essentially based on the observation of the stars.

In IT, the ISO 8601 and RFC 3339 standards are the widely accepted guidelines.

ISO 8601
Data elements and interchange formats
– Information interchange –
Representation of dates and times

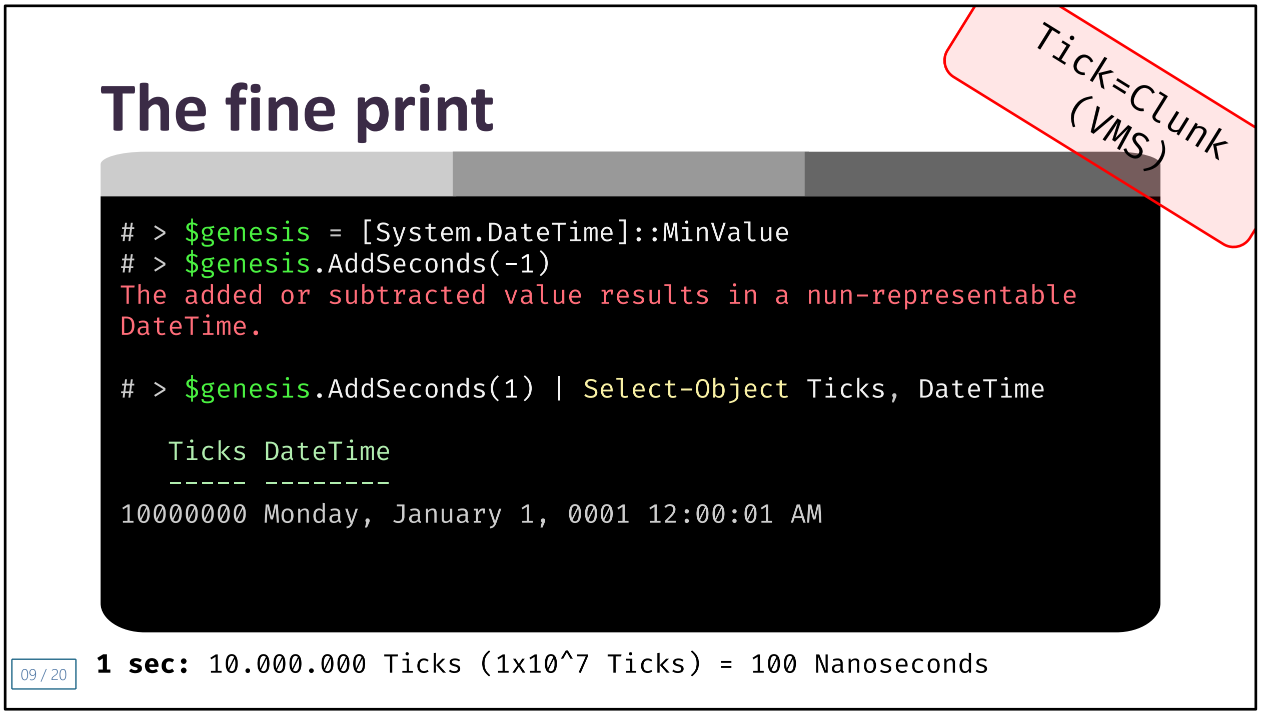
Getting a properly formatted timestamp (for example: 2024-06-15T20:23:28.2222591+05:45)
`Get-Date -Format O`

RFC 3339
Date and Time on the Internet: Timestamps

Getting a properly formatted timestamp
`Get-Date -AsUTC -Format r`

Example: Sat, 15 Jun 2024 14:38:28 GMT)

Both standards differ, but largely overlap. The following link provides a very good overview utilizing a diagram. Highly recommended:
https://ijmacd.github.io/rfc3339-iso8601/



The fine print

```
# > $genesis = [System.DateTime]::MinValue
# > $genesis.AddSeconds(-1)
The added or subtracted value results in a nun-representable
DateTime.

# > $genesis.AddSeconds(1) | Select-Object Ticks, DateTime

   Ticks DateTime
   ----- --------
10000000 Monday, January 1, 0001 12:00:01 AM
```

1 sec: `10.000.000 Ticks (1x10^7 Ticks) = 100 Nanoseconds`

The case of the missing prefix
Since there is no SI (**I**ntenational **S**ystem of Units) prefix for `10^7`, we refer to it with the next available prefix, wich is nano seconds representing `10^9`
(10 to the power of 9).
https://en.wikipedia.org/wiki/Metric_prefix

Some people proposed "hectonanoseconds", wich is not common.
https://stackoverflow.com/questions/43009885/is-there-a-standard-term-for-windowss-100-nanosecond-time-unit

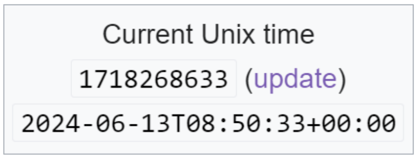
David Cutler also invented VMS and Windows NT. Both use the same interval for time measurement: 100 ns, also called Ticks (Windows NT) or Clunks (VMS)
https://static.hlt.bme.hu/semantics/external/pages/nagyszám-aritmetika/en.wikipedia.org/wiki/VAX/VMS.html

Why do we use 100 ns, why not 10 ns or just 1 ns?
Lowering the interval would results in big numbers that exceed the size of a 64 bit integer.
This will also be true if you use the "Win32Epoch" as the starting point.

```
$apocalypse = [System.DateTime]::MaxValue
$apocalypse.Ticks.GetType().FullName        ## 64 bit integer
($apocalypse.Ticks * 10).GetType().FullName  ## System.Double

$win32EpochBirth=[System.DateTime]::new(1961,1,1,0,0,0,[DateTimeKind]::UTC)
(($apocalypse.Ticks - $win32EpochBirth.Ticks) * 10).GetType().FullName
```

UNIX era

UNIX epoch: Jan 1, 1970


Current Unix time

1718268633 (update)

2024-06-13T08:50:33+00:00

https://en.wikipedia.org/wiki/Unix_time

10 / 20

Unix time
"Unix time[a] is a date and time representation widely used in computing. It measures time by the number of non-leap seconds that have elapsed since 00:00:00 UTC on 1 January 1970, the Unix epoch. In modern computing, values are sometimes stored with higher granularity, such as microseconds or nanoseconds.

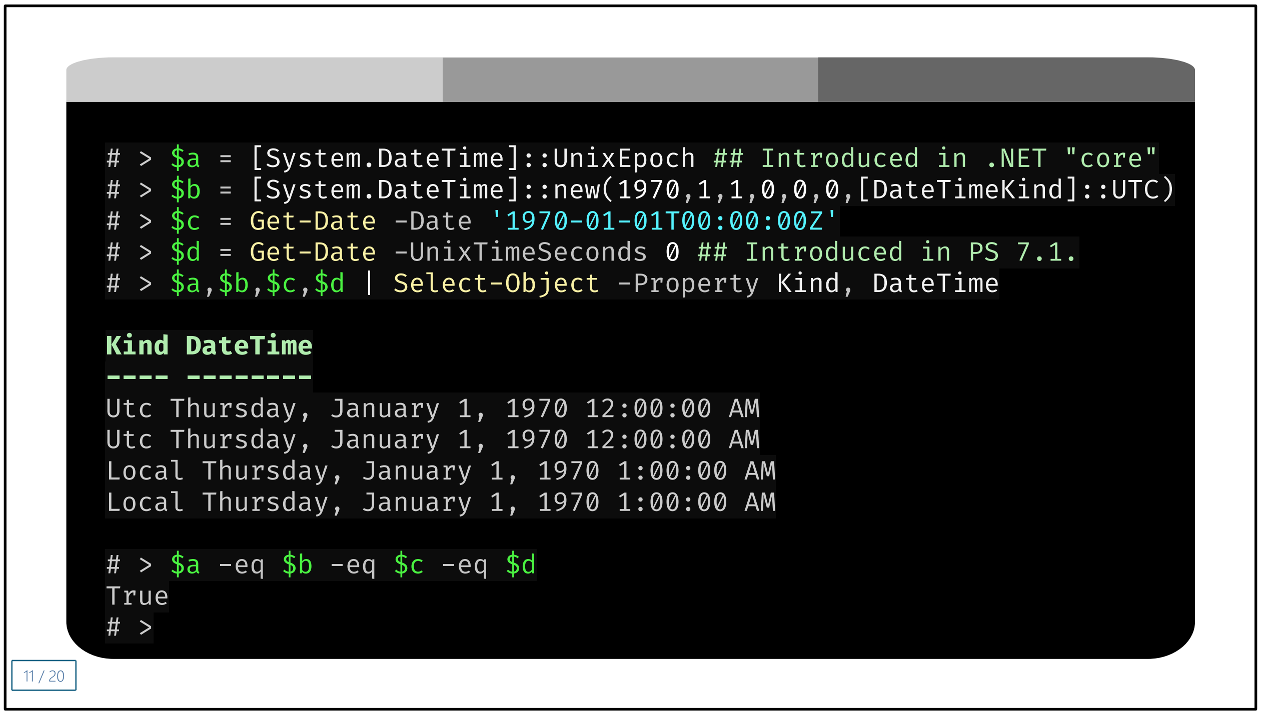
Unix time originated as the system time of Unix operating systems. It has come to be widely used in other computer operating systems, file systems, programming languages, and databases."
https://en.wikipedia.org/wiki/Unix_time

Big Picture
Dennis Ritchie and Ken Thompson, creators of UNIX and C, with a PDP-11/20.
https://www.computerhistory.org/collections/catalog/102685442

Dos and Donts
Since "Get-Date" does not work in a correct, reliable and predictable way throughout differenct PowerShell-Versions, I highly recommend to use the framework instead of the cmdlet to get the UNIX epoch timestamp.

Do
```
[System.DateTime]::new(1970,1,1,0,0,0,[DateTimeKind]::UTC)
```

Dont
```
Get-Date -UFormat %s
```

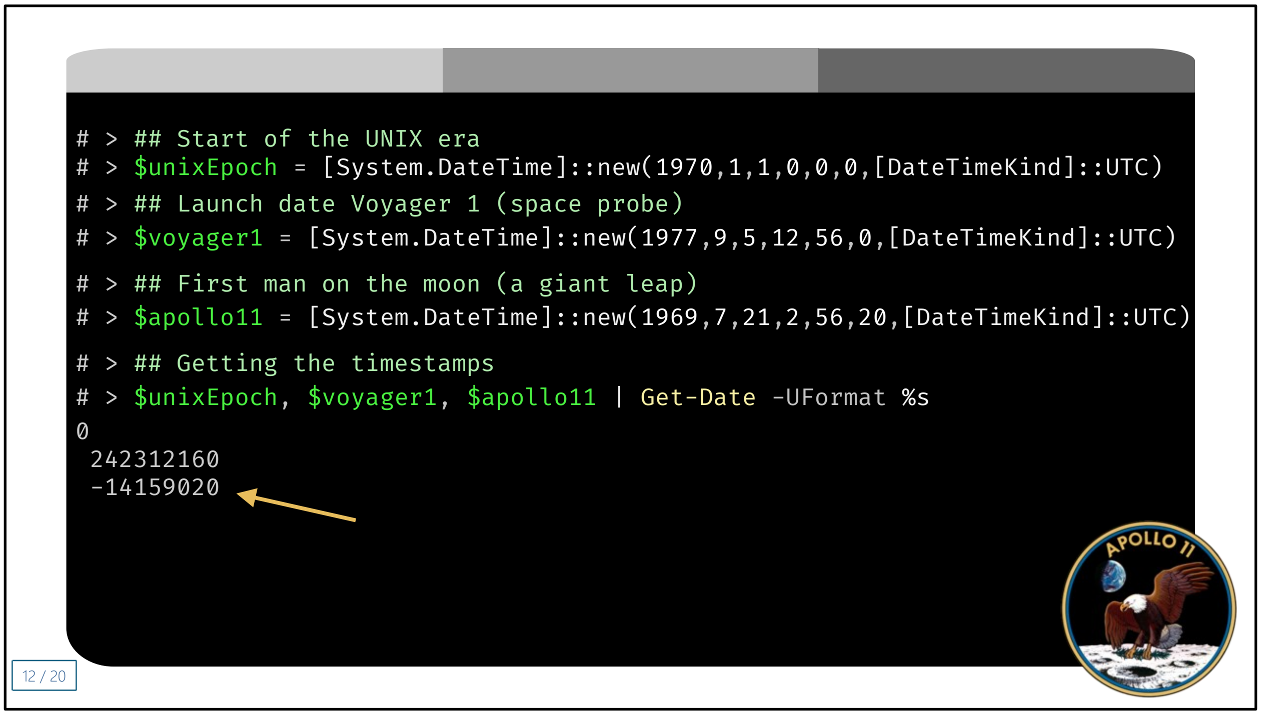


```
# > $a = [System.DateTime]::UnixEpoch ## Introduced in .NET "core"
# > $b = [System.DateTime]::new(1970,1,1,0,0,0,[DateTimeKind]::UTC)
# > $c = Get-Date -Date '1970-01-01T00:00:00Z'
# > $d = Get-Date -UnixTimeSeconds 0 ## Introduced in PS 7.1.
# > $a,$b,$c,$d | Select-Object -Property Kind, DateTime

Kind DateTime
---- --------
Utc Thursday, January 1, 1970 12:00:00 AM
Utc Thursday, January 1, 1970 12:00:00 AM
Local Thursday, January 1, 1970 1:00:00 AM
Local Thursday, January 1, 1970 1:00:00 AM

# > $a -eq $b -eq $c -eq $d
True
# >
```

11 / 20

The problem with "Get-Date"
The Get-Date cmdlet will honor the current local time zone, which will lead to "unexpected" results. If you change the TZ, you will "see" different dates – the UNIX epoch is translated to your local time. If you want to avoid this, use "[System.DateTime]", where you can enforce UTC output.

```
$a = [System.DateTime]::UnixEpoch  ## Introduced in .NET "core"
$b = [System.DateTime]::new(1970,1,1,0,0,0,[DateTimeKind]::UTC)
$c = Get-Date -Date '1970-01-01T00:00:00Z'
$d = Get-Date -UnixTimeSeconds 0   ## Introduced in PS 7.1.

## This will be true, all timestamps point to the UNIX epoch
$a -eq $b -eq $c -eq $d


## With TZ "Nepal Standard Time"
$a,$b,$c,$d | Select-Object -Property Kind,DateTime

 Kind DateTime
 ---- --------
  Utc Thursday, January 1, 1970 12:00:00 AM
  Utc Thursday, January 1, 1970 12:00:00 AM
Local Thursday, January 1, 1970 5:45:00 AM
Local Thursday, January 1, 1970 5:45:00 AM
```



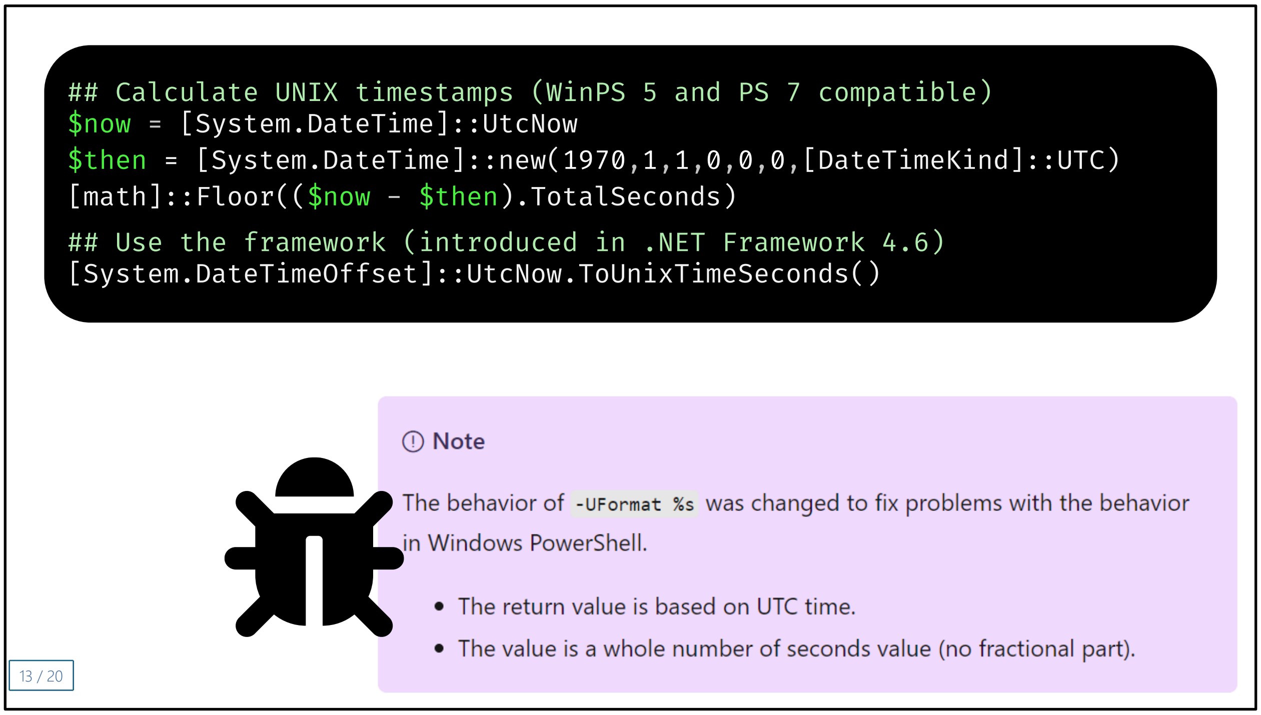
```
# > ## Start of the UNIX era
# > $unixEpoch = [System.DateTime]::new(1970,1,1,0,0,0,[DateTimeKind]::UTC)
# > ## Launch date Voyager 1 (space probe)
# > $voyager1 = [System.DateTime]::new(1977,9,5,12,56,0,[DateTimeKind]::UTC)
# > ## First man on the moon (a giant leap)
# > $apollo11 = [System.DateTime]::new(1969,7,21,2,56,20,[DateTimeKind]::UTC)
# > ## Getting the timestamps
# > $unixEpoch, $voyager1, $apollo11 | Get-Date -UFormat %s
0
 242312160
 -14159020
```

This code above supports Windows PowerShell 5 and PowerShell 7. Pay attention to the fact that UNIX timestamps always refer to UTC.

"**Voyager 1** is a space probe launched by NASA on September 5, 1977, as part of the Voyager program to study the outer Solar System and the interstellar space beyond the Sun's heliosphere. It was launched 16 days after its twin, Voyager 2. It communicates through the NASA Deep Space Network (DSN) to receive routine commands and to transmit data to Earth."
https://en.wikipedia.org/wiki/Voyager_1

"**Apollo 11** (July 16–24, 1969) was the American spaceflight that first landed humans on the Moon. Commander Neil Armstrong and Lunar Module Pilot Buzz Aldrin landed the Apollo Lunar Module Eagle on July 20, 1969, at 20:17 UTC, and Armstrong became the first person to step onto the Moon's surface six hours and 39 minutes later, on July 21 at 02:56 UTC."
https://en.wikipedia.org/wiki/Apollo_11

Picture: Apollo 11 patch
http://history.nasa.gov/patches/Apollo/Apollo11.jpg



```
## Calculate UNIX timestamps (WinPS 5 and PS 7 compatible)
$now = [System.DateTime]::UtcNow
$then = [System.DateTime]::new(1970,1,1,0,0,0,[DateTimeKind]::UTC)
[math]::Floor(($now - $then).TotalSeconds)

## Use the framework (introduced in .NET Framework 4.6)
[System.DateTimeOffset]::UtcNow.ToUnixTimeSeconds()
```

ⓘ **Note**

The behavior of `-UFormat %s` was changed to fix problems with the behavior in Windows PowerShell.

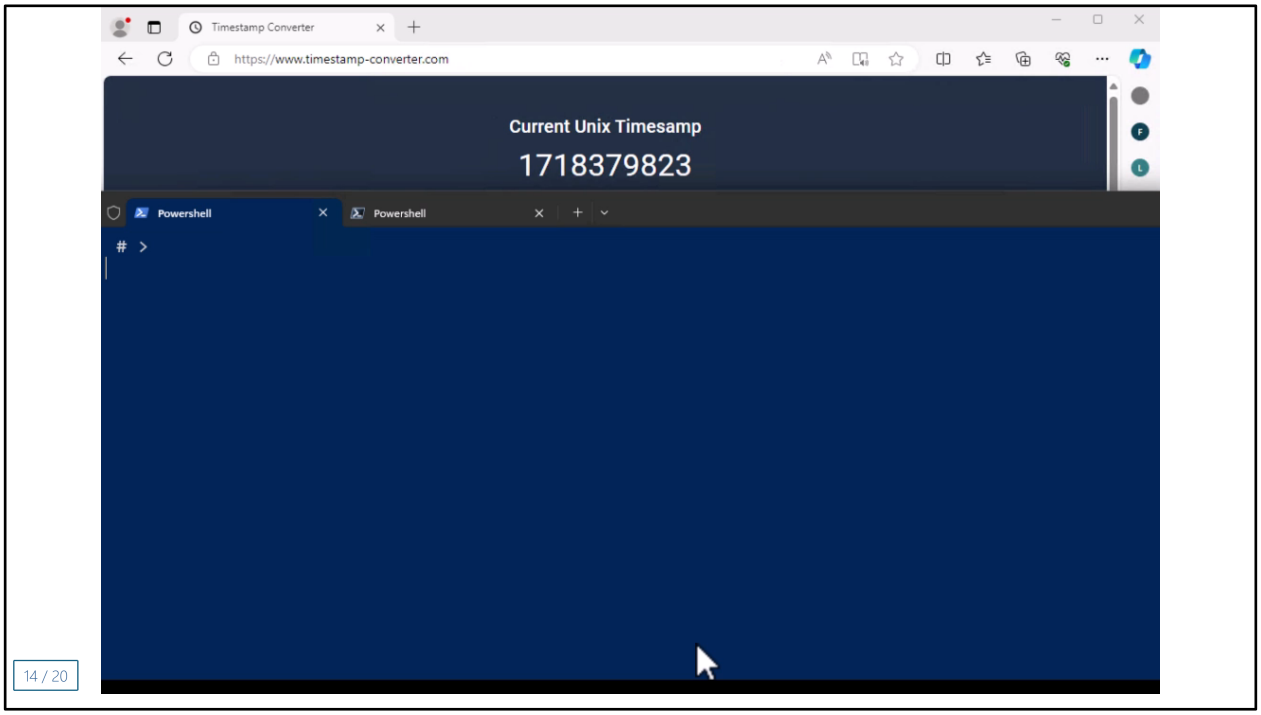
- The return value is based on UTC time.
- The value is a whole number of seconds value (no fractional part).

13 / 20

Using "**Get-Date -UFormat %s**", as seen on the previous slide, can be confusing. Despite the fact that Get-Date honors the local time, different time zone settings must always lead to the same UNIX timestamp, since these are defined as UTC.

Windows PowerShell 5 incorrectly includes the local time offset, so that different results are obtained when the time zone is changed. Furthermore, Windows PowerShell 5 outputs decimal places, which is also questionable. These errors have been fixed in PowerShell 7 (see Note).

UTC is the abbreviation for "Coordinated Universal Time", also know as GMT, "Greenwich Mean Time".



```
# >
```

Try the demo code yourself. Compare the results in Windows PowerShell 5 and PowerShell 7.

```powershell
## Demo code
Set-TimeZone -Id 'UTC'
$resultUTC = Get-Date -UFormat %s

Set-TimeZone -Id 'Nepal Standard Time'
$resultNepal = Get-Date -UFormat %s

$resultUTC, $resultNepal
```

Win32 era

Why is the Win32 epoch January 1, 1601?

The FILETIME structure records time in the form of 100-nanosecond intervals since January 1, 1601. Why was that date chosen?

The Gregorian calendar operates on a 400-year cycle, and 1601 is the first year of the cycle that was active at the time Windows NT was being designed. In other words, it was chosen to make the math come out nicely.

I actually have the email from Dave Cutler confirming this.

Raymond Chen
Follow

https://devblogs.microsoft.com/oldnewthing/20090306-00/?p=18913

15 / 20

Raymond Chen
This is the complete announcement from the blog.

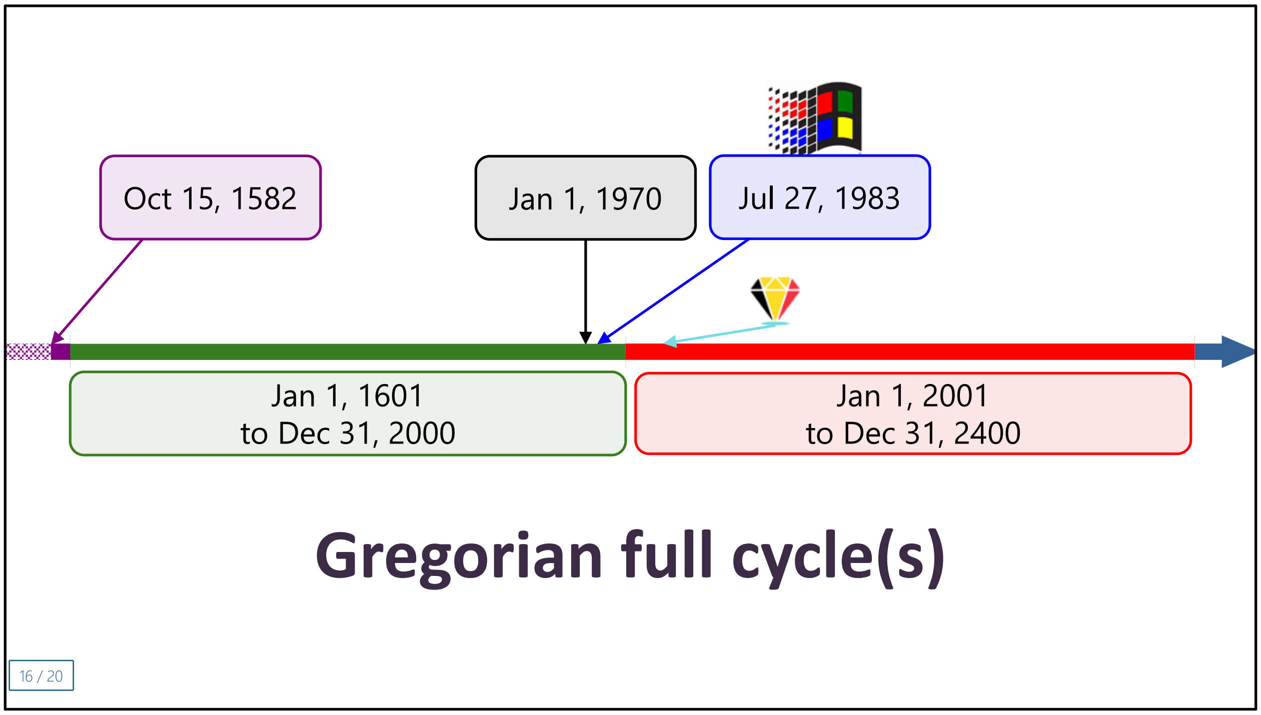
"Public Service Announcement: This weekend marks the start of Daylight Saving Time in most parts of the United States.

The FILETIME structure records time in the form of 100-nanosecond intervals since January 1, 1601. Why was that date chosen?

The Gregorian calendar operates on a 400-year cycle, and 1601 is the first year of the cycle that was active at the time Windows NT was being designed. In other words, it was chosen to make the math come out nicely.

I actually have the email from Dave Cutler confirming this."

https://devblogs.microsoft.com/oldnewthing/20090306-00/?p=18913



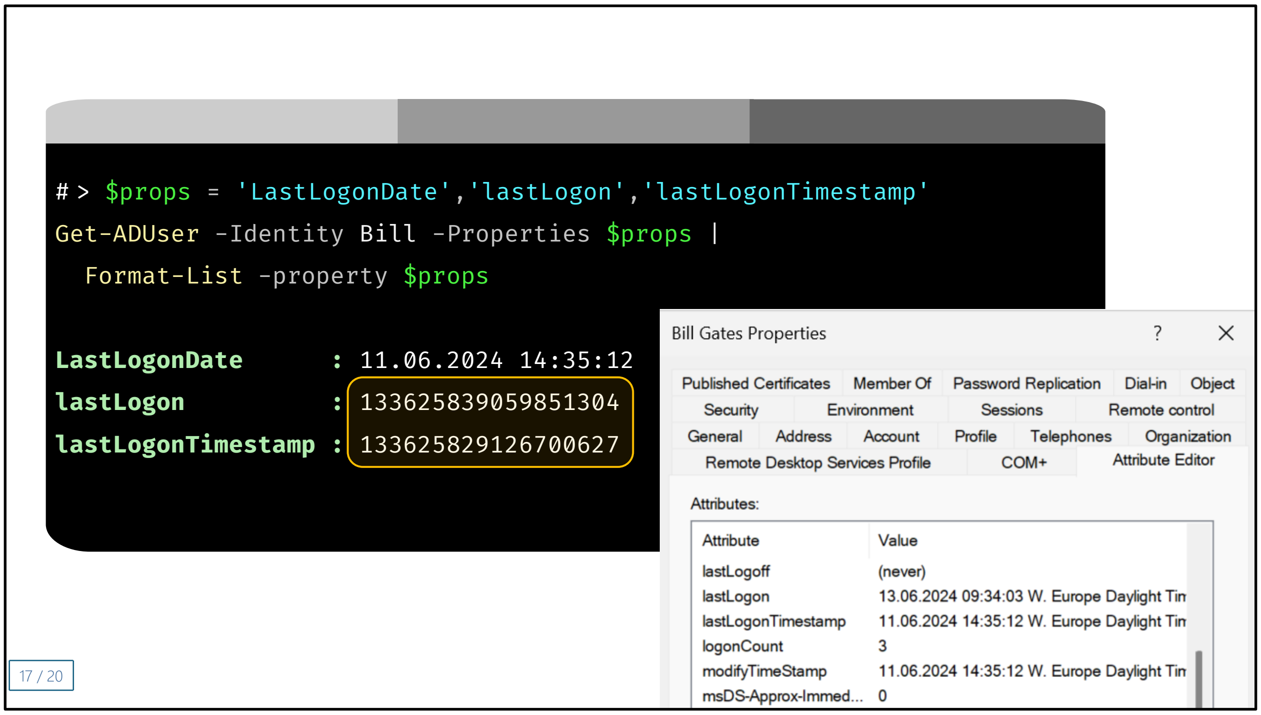
Gregorian full cycle(s)

With the fixes of Pope Gregory, a complete calendar cycle consists of 400 years. This can also be described as **a solar cycle**.

"The solar cycle is a 28-year cycle of the Julian calendar, **and 400-year cycle of the Gregorian calendar** with respect to the week. It occurs because leap years occur every 4 years, typically observed by adding a day to the month of February, making it February 29th. There are 7 possible days to start a leap year, making a 28-year sequence.

This cycle also occurs in the Gregorian calendar, but it is interrupted by years such as 1700, 1800, 1900, 2100, 2200, 2300, 2500, 2600, 2700, 2900, and any year that is divisible by 100, but not by 400. These years are common years and are not leap years. This interruption has the effect of skipping 16 years of the solar cycle between February 28 and March 1. Because the Gregorian cycle of 400 years has exactly 146,097 days, i.e. exactly 20,871 weeks, one can say that the Gregorian so-called solar cycle lasts 400 years."

https://en.wikipedia.org/wiki/Solar_cycle_(calendar)



```
# > $props = 'LastLogonDate','lastLogon','lastLogonTimestamp'
Get-ADUser -Identity Bill -Properties $props |
  Format-List -property $props

LastLogonDate      : 11.06.2024 14:35:12
lastLogon          : 133625839059851304
lastLogonTimestamp : 133625829126700627
```

Bill Gates Properties

```
Attribute             Value
lastLogoff            (never)
lastLogon             13.06.2024 09:34:03 W. Europe Daylight Tin
lastLogonTimestamp    11.06.2024 14:35:12 W. Europe Daylight Tin
logonCount            3
modifyTimeStamp       11.06.2024 14:35:12 W. Europe Daylight Tin
msDS-Approx-Immed...  0
```

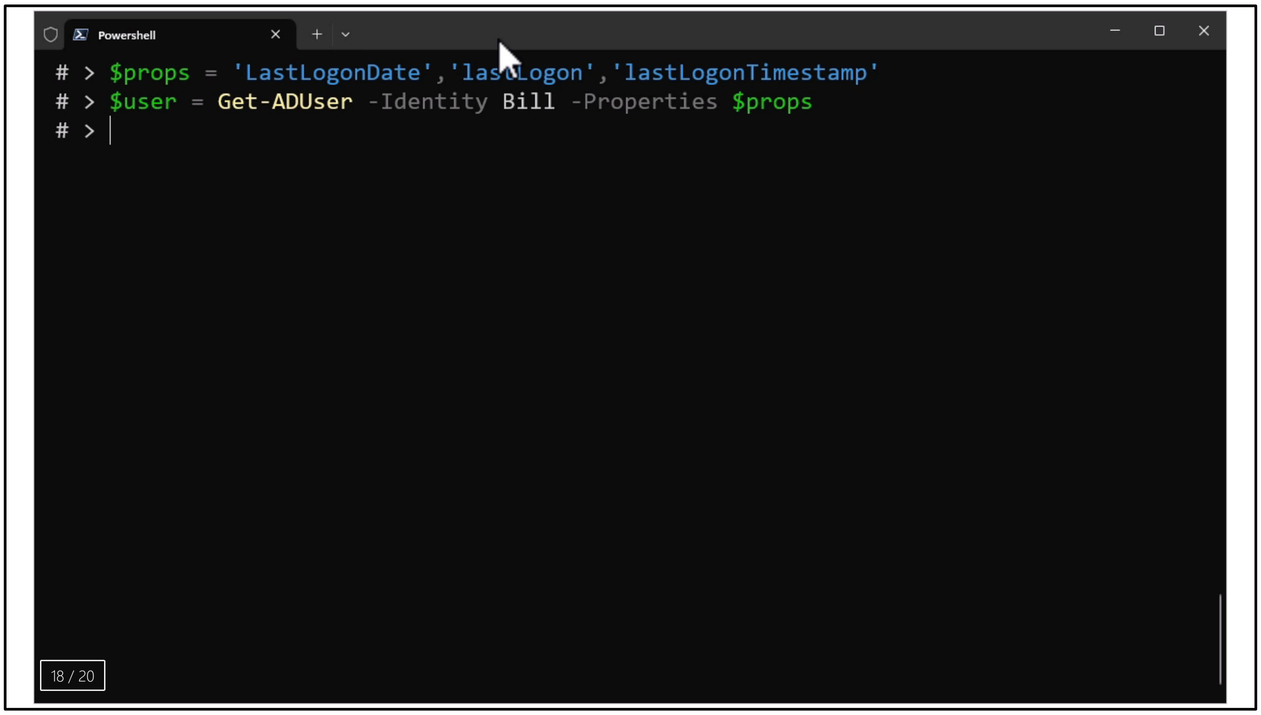
17 / 20

(a) LastLogonDate
"PowerShell was nice enough to give us a third option to query by. LastLogonDate is a converted version of LastLogontimestamp. "
Actually, "LastLogonDate" should have been named "LastLogonTimeStampDate" since PowerShell converts the lastLogonTimeStamp (c) in a readable format as DateTime object.

(b) lastLogon
"This attribute is not replicated across domain controllers. It is precise but requires querying each domain controller."

(c) lastLogonTimeStamp
This attribute is replicated but not in real-time. It is designed to help identify inactive accounts and typically has a replication latency of up to 14 days to reduce replication traffic. Therefore, it is not always up-to-date to the exact time of the last logon.

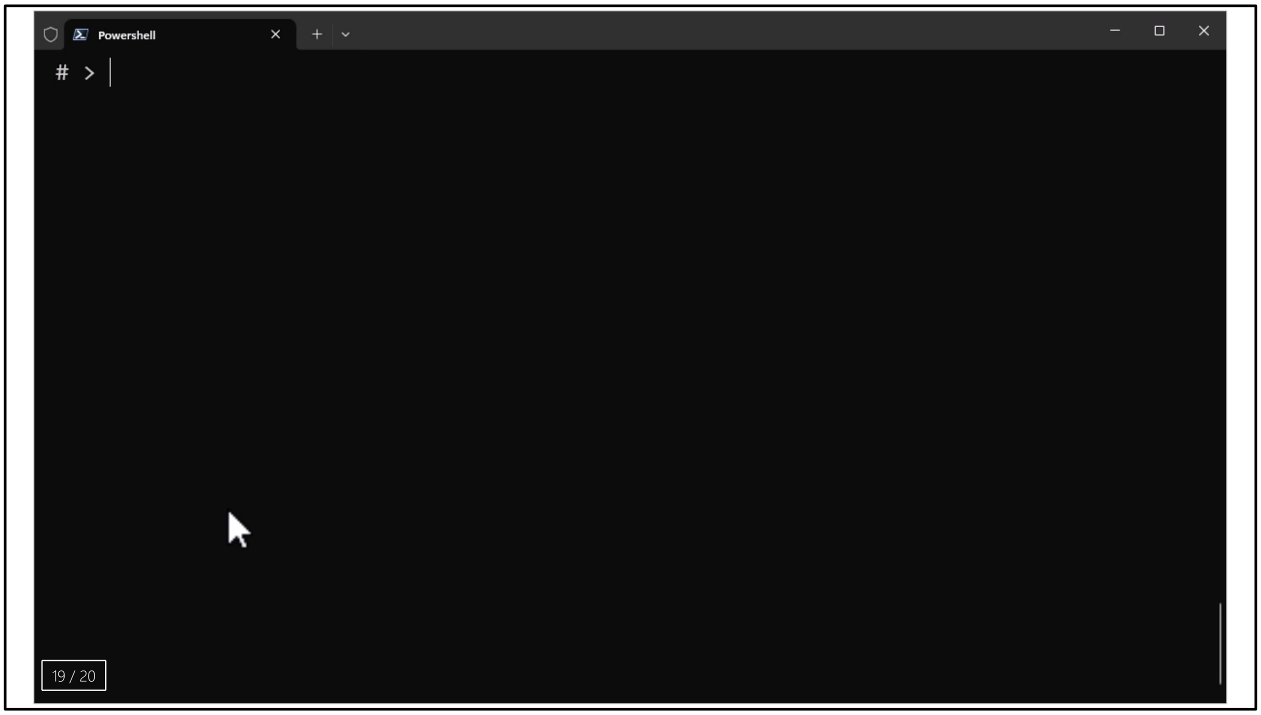
https://learn.microsoft.com/en-us/archive/technet-wiki/22461.understanding-the-ad-account-attributes-lastlogon-lastlogontimestamp-and-lastlogondate



```
# > $props = 'LastLogonDate','lastLogon','lastLogonTimestamp'
# > $user = Get-ADUser -Identity Bill -Properties $props
# >
```

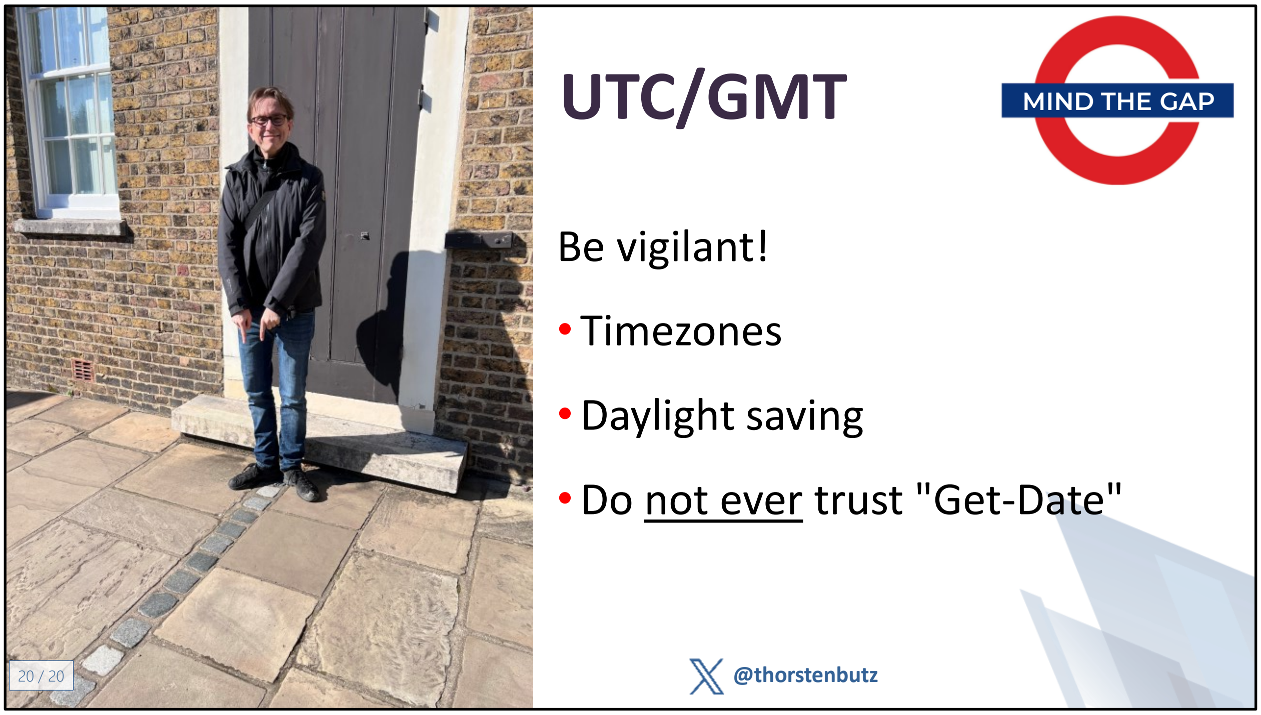
```
$props = 'LastLogonDate','lastLogon','lastLogonTimestamp'
$user = Get-ADUser -Identity Bill -Properties $props

$objLastLogon = [System.DateTime]::new($user.lastlogon,'UTC')
$objLastLogonTimestamp =
[System.DateTime]::new($user.lastlogonTimeStamp,'UTC')

$objLastLogon           ## Thursday, June 13, 0424, 7:34:03 AM
$objLastLogonTimestamp  ## Tuesday, June 11, 0424, 12:35:12 PM
```

It is obvious that these results cannot be correct. The year 424 is out of our scope.
The reason is, that the ticks from Active Directory are also intervals of 100 ns, but the AD uses another epoch – not the year 1, as DateTime objects do.



```
$props = 'LastLogonDate','lastLogon','lastLogonTimestamp'
$user = Get-ADUser -Identity Bill -Properties $props

$objLastLogon = [System.DateTime]::new($user.lastlogon,'UTC')
$objLastLogonTimestamp = [System.DateTime]::new($user.lastlogonTimeStamp,'UTC')


$objLastLogon.AddYears(1600) ## Thursday, June 13, 2024, 7:34:03 AM
$objLastLogonTimestamp.AddYears(1600) ## Tuesday, June 11, 2024, 12:35:12 PM
```

Adding 1600 years corrects the results.
The Win32 epoch differs from the ISO 8601 / common era this way.



UTC/GMT

MIND THE GAP

Be vigilant!

- Timezones
- Daylight saving
- Do not ever trust "Get-Date"

@thorstenbutz

20 / 20

Summary
Be careful in terms of timezones and daylight saving. Check your code with different local TZ configurations. If you code with UTC as the locally configured TZ, be aware of the risks when your code runs on computers with different settings.

As a general advice: be very careful with Get-Date – or even better, avoid it.
Use DateTime objects instead and enforce UTC timestamps where needed.

Me standing on the prime meridian
The picture was taken at the Royal Observatory in Greenwich, London.
The cobblestones beneath my feet show the course of the prime meridian (00° 00' 00").

About this talk

Venue:	PSConfEU 2024
Speaker:	Thorsten Butz
Title:	about_Epochs
Format:	PoShaKucha
Twitter:	@thorstenbutz
Homepage:	thorsten-butz.de
Podcast:	slidingwindows.de

Code repos
https://github.com/thorstenbutz/PoShaKucha
https://github.com/psconfeu/2024

v24.06.22