Why is every other line of my file mysteriously changed to nonsense Chinese characters?

devblogs.microsoft.com/oldnewthing/20220328-00

March 28, 2022

Raymond Chen

A customer found that every other line of their file mysteriously changed to nonsense Chinese characters:

```
// Microsoft Visual C++ generated resource script.
∅椀渀挀氀甀搀攀 ∀爀攀猀漀甀爀挀攀⸀栀
#define APSTUDIO_READONLY_SYMBOLS
—— 䜀攀渀攀爀愀琀攀搀 昀爀漀洀 琀栀攀 吀䔀堀吀䤀一䌀䰀唀䐀䔀 爀攀猀漀甀爀挀攀
```

My goodness, that does indeed look mysterious.

But it's less mysterious if you look at it in a hex editor:

```
FF FE 2F 00 2F 00 4D 00-69 00 63 00 72 00 6F 00  ../././M.i.c.r.o.
73 00 6F 00 66 00 74 00-20 00 56 00 69 00 73 00  s.o.f.t. .V.i.s.
75 00 61 00 6C 00 20 00-43 00 2B 00 2B 00 20 00  u.a.l. .C.+.+. .
67 00 65 00 6E 00 65 00-72 00 61 00 74 00 65 00  g.e.n.e.r.a.t.e.
64 00 20 00 72 00 65 00-73 00 6F 00 75 00 72 00  d. .r.e.s.o.u.r.
63 00 65 00 20 00 73 00-63 00 72 00 69 00 70 00  c.e. .s.c.r.i.p.
74 00 2E 00 0D 00 0D 0A-00 23 00 69 00 6E 00 63  t........#.i.n.c
00 6C 00 75 00 64 00 65-00 20 00 22 00 72 00 65  .l.u.d.e. .".r.e
00 73 00 6F 00 75 00 72-00 63 00 65 00 2E 00 68  .s.o.u.r.c.e...h
00 22 00 0D 00 0D 0A 00-23 00 64 00 65 00 66 00  .".......#.d.e.f.
69 00 6E 00 65 00 20 00-41 00 50 00 53 00 54 00  i.n.e. .A.P.S.T.
55 00 44 00 49 00 4F 00-5F 00 52 00 45 00 41 00  U.D.I.O._.R.E.A.
44 00 4F 00 4E 00 4C 00-59 00 5F 00 53 00 59 00  D.O.N.L.Y._.S.Y.
4D 00 42 00 4F 00 4C 00-53 00 0D 00 0D 0A 00 2F  M.B.O.L.S......./
00 2F 00 20 00 47 00 65-00 6E 00 65 00 72 00 61  ./. .G.e.n.e.r.a
00 74 00 65 00 64 00 20-00 66 00 72 00 6F 00 6D  .t.e.d. .f.r.o.m
00 20 00 74 00 68 00 65-00 20 00 54 00 45 00 58  . .t.h.e. .T.E.X
00 54 00 49 00 4E 00 43-00 4C 00 55 00 44 00 45  .T.I.N.C.L.U.D.E
00 20 00 72 00 65 00 73-00 6F 00 75 00 72 00 63  . .r.e.s.o.u.r.c
00 65 00 2E 00 0D 00 0D-0A 00 2F 00 2F 00 0D 00  .e........././...
```

The first few bytes of the file are clearly recognizable as encoded in UTF-16LE, complete with U+FEFF BOM.

And then something weird happens at the end of the first line of text: You would expect the line to end with the byte sequence `0D 00 0A 00`, corresponding to the UTF-16LE encoding of U+000D CARRIAGE RETURN followed by U+000A LINE FEED. But instead, there's this extra `0D` byte stuck into the stream before the `0A`, and that messes things up.

Since UTF-16LE encoding uses pairs of bytes, inserting a single byte throws off the synchronization. All the even bytes become odd, and all odd bytes become even. This is why the second line (outlined) comes out as Chinese nonsense: Putting ASCII alphabetics in the high-order byte of UTF-16 code units results in nonsense Chinese characters.

When we get to the end of the (now garbled) line, again there is a mystery `0D` interloper byte immediately before the `0A`. This second unwanted byte restores parity, which means that the bytes of the third line of text are properly paired again, and they appear normal.

And then when we get to the end of the third line, the cycle repeats, with a rogue `0D` throwing off the synchronization and causing the fourth line to become gibberish. This continues for the remainder of the file: The corrupted line ending causes every other line to turn into nonsense.

Okay, so now we know what's going on, but how did the file get corrupted in this way?

The insertion of `0D` before `0A` is coming from CRLF/LF conversion. Windows text files end with carriage return and line feed, but Unix text files end with just line feed. The usual algorithm for converting from Windows format to Unix format is to remove carriage returns if they are immediately followed by a line feed. And the usual algorithm for converting from Unix format to Windows format is to insert carriage returns before every line feed.

Of course, if you "insert" and "remove" the carriage return, you have to do so with the correct encoding.

What happened here is that the original file was encoded in UTF-16LE and used Windows line endings. My guess is that this file was then stored in a system that uses Unix line endings, so the line endings were converted, but the conversion code interprets the file as UTF-8, which encodes a CRLF as the byte sequence `0D 0A`. That sequence does not occur in the file, so the conversion is a nop, and the file goes into storage unchanged.

Some time later, the file is retrieved from storage, and the line endings are now converted from Unix-style to Windows-style. Again, the conversion code assumes the file is UTF-8 encoded, so it inserts a carriage return `0D` before every line feed `0A`.

And that is why the `0A` bytes are being inserted and messing up the file.

There is a German word for this: *verschlimmbessern*: To make something worse in a well-intentioned but failed attempt to improve it.

A common place you'll see this type of corruption is when you ask Visual Studio to create a new project with a git repo.

Some Visual Studio project templates create source files encoded as UTF-16LE, and it's common to add a `.gitattributes` file that says that source files are `text eol=crlf`. If you do that, then git follows your instructions and converts files from Windows line endings to Unix line endings when committing, and performs the reverse conversion when checking files out.

However, git assumes text files are encoded as UTF-8.

To avoid *Verschlimmbesserung*, you should re-encode your UTF-16LE files as UTF-8 before adding them to the git repo. Note that Resource Compiler *.rc files default to the ANSI code page, even if the file begins with a UTF-8 BOM. You need to explicitly inform the Resource Compiler that the file is encoded as UTF-8 by saying

```
#pragma code_page(65001) // UTF-8
```

Raymond Chen

Follow