

Opposition to and Comment on L2/23-107

Fredrick R. Brennan <copypaste@kittens.ph>

June 30, 2023

1 Abstract

Microsoft submitted at Unicode Technical Committee № 175 (25th–27th April 2023) a document entitled “Proper Complex Script Support in Text Terminals”.¹ *While I certainly agree with the need for standardization in this area*, and furthermore support the creation of the new Unicode project therein proposed, the Terminal Complex Script Support Working Group (TCSS WG), I believe Microsoft has failed to take into account existing standards and the way they propose to do text shaping in text terminals is fundamentally flawed.

2 The Actual State of the Art

Microsoft cited only three examples of text terminals in its document:

1. An IBM 5550 displaying mixed Japanese/Latin text *circa* 1980;
2. Windows Terminal (`cmd.exe`)
3. Apple `Terminal.app`

None of these examples adequately reflect the state of the art. Text terminal development primarily happens in the free software community, of which Microsoft is only occasionally a member. This is not a slight against Microsoft, merely a statement of fact that they are unlikely to be aware of the true state of the art due to their separation from the spaces it has been developed in.

In actuality, it is already very possible to display properly shaped OpenType text in text terminals (also, and more correctly, known as “terminal emulators”) in the two most common terminal emulators found on GNU/Linux desktops:² GNOME Terminal and Konsole. I believe the true current state of the art to be best reflected in `m1term` (the **m**ultilingual **t**erminal). Below, the resulting output to the pseudo-`tty` the `motd`³ of my FTP server⁴ is displayed in all three, from left to right.

¹Li, Renzhi; Howett, Dustin; Constable, Peter (2023). *Proper Complex Script Support in Text Terminals*. Unicode Technical Committee Meeting № 175. L2/23-107.

²And other desktops of similar pedigree such as FreeBSD desktops, Haiku OS desktops, *et cetera*.

³Message of the Day

⁴<https://デブ.狸.agency>; although for the convenience of readers it is also both attached to this document and available from <https://debu.tanuki.agency>.

2.1 デブ.狸.agency motd as displayed on GNU/Linux

While as yet imperfect, all three of these terminals do a far better job than either Windows Terminal or Terminal.app:

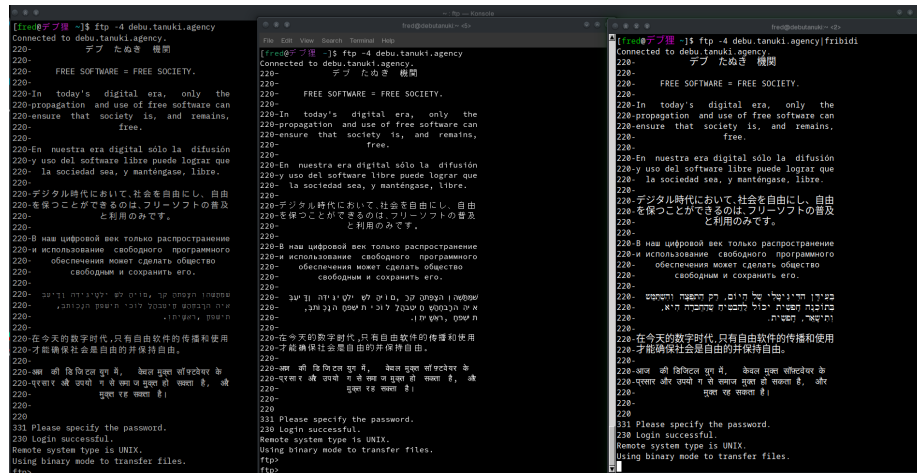


Figure 1: From left: Konsole 23.04.2, GNOME Terminal 3.48.1⁷, and mlterm version 3.9.3⁸.

I will in the following section attempt to explain in brief (please excuse any errors) how this works.

3 The free software text stack

In free software operating systems, the text stack is essentially thus when complex shaping is desired:

FreeType

Responsible for rasterization only.

Fribidi

Implements Unicode’s bi-directional algorithm.

ICU

Implements Unicode’s linebreaking algorithm.

HarfBuzz

Moves the bitmaps produced by FreeType into position according to rules defined either in OpenType Layout, Graphite, or Apple Advanced Typography (AAT) rules in a way that is cognizant of Fribidi’s results.

Fontconfig

Chooses an appropriate font for a given TUS script.

⁷using VTE 0.72.2 +BIDI +GNUTLS +ICU +SYSTEMD

⁸post/2023-06-11, using features: otl ssh implugin imagelib(builtin) utmp

Pango⁹

Does the actual rendering, and is responsible for linebreaking, which HarfBuzz cannot do on its own.¹⁰

1. Calls Fontconfig multiple times to find fonts to display a given string.
2. Loads those fonts with FreeType, ignoring unloadable fonts.
3. Shapes the strings with HarfBuzz.
4. Returns either a vector (e.g. PDF output) or a raster.

Immediately a problem emerges, which Microsoft’s authors are right to point out: Pango assumes it does not need to fit text into grid spaces, and can return a raster of any width, or, indeed, zero-width.

Two standards that I’m aware of exist to address this.

4 ECMA TR-53

The European Computer Manufacturer’s Association in June of 1993 released Technical Report № 53 entitled “Handling of Bi-Directional Texts”.¹¹

It is curious that this paper is not mentioned by the Microsoft authors. Nevertheless, it has a lot of issues; it predates Unicode BIDI, and it may indeed be unimplementable according to the author of `libvte`.¹²

5 Freedesktop BiDiTE WG Recommendation

Anton Kochkov’s implementation is by far the most robust and he has put the most thought into the problem of any free software developer as far as I can tell.

His is the (still draft) recommendation of Freedesktop.org’s BiDi in Terminal Emulators (BiDiTE) WG.¹³

6 Incompatibilities

A large oversight is noticed immediately in Microsoft’s document in that line-breaking is to be figured out at a “later time”. Certainly this is why they have invited a WG. However, the whole way that Microsoft intends that complex shaping and BiDi be done, with a “working area” for each word, is both incompatible and likely to lead to issues.

In my opinion the most logical process looks something more like:¹⁴

1. Divide the input into words;

⁹Often combined with Cairo as Pangocairo.

¹⁰See [What HarfBuzz doesn’t do](#).

¹¹https://www.ecma-international.org/wp-content/uploads/ECMA_TR-53_2nd_edition_june_1992.pdf

¹²Kochkov, Anton (@XVilka). [Review of TR-53](#). Freedesktop.org BiDi in Terminal Emulators Working Group.

¹³Kochkov, Anton. <https://terminal-wg.pages.freedesktop.org/bidi/>

¹⁴Though much like Microsoft’s document, this proposal is nowhere near final.

2. Run the Unicode BiDi and line-breaking algorithms;
3. Run `wcswidth()` on all the words or word segments in the case of a hyphenated word;
4. Measure the text's width and fit it into the minimum number of spaces, or else if it could be stretched or compressed $\pm 5\%$ to fit into the number of text spaces, do so;
5. Recommend monospace fonts be used wherever possible;
6. Keep the existing norm that CJK characters are equal to two grid spaces without exception.

I look forward to discussing these issues at the TCSS WG.