

File analysis tool

John Taylor (onefouronefour limited) 2015-07-30

Years ago I wrote a simple utility to analyse a file into its byte components. I find it very useful and always bring it to each new system I work on, together with some elementary test files; for example a file '256' containing 256 bytes, one of each bit pattern. Where necessary I have rewritten it in C, Java, Perl and awk (making use of od to scan the file to accommodate null characters, which awk cannot handle properly).

As a first examination of a file, it easy to tell whether a text file is DOS- or Unix-formatted by looking at the counts of new line (0x0a) and carriage return (0x0d) characters. Often, problems are resolved by checking whether a file contains tabs, nulls, delete and other unprintables. In text files, characters above 0x7f may represent accented characters or other special values.

However, nowadays many files are encoded in UTF-8 (and most web pages too). Simply adding counts of all bit patterns is not sufficient to understand the large “alphabet” of Unicode characters which are represented in 1, 2, 3 or 4 bytes.

I have rewritten the program again, now in python (utfa.py), this time including support for all UTF-8 characters and adding a mechanism to read web pages. Since the two widely used versions of python (2.x and 3.x) handle unicode differently, I wrote the program to detect which version of python is in use and to handle unicode characters appropriately.

By default, the program assumes files/web pages are UTF-8 encoded. If the input cannot be decoded correctly it suggests two alternatives to handle this.

To illustrate, here is the same '256' test file:

```
utfa.py 256
file 256 is not valid utf-8, try analysing file as bytes using flag -b or enable error replacement with flag -e
```

using the first suggestion:

```
utfa.py 256 -b
```

byte analysis for 256

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	-- characters --
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	!"#\$%&'()*+,-./
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0123456789:;<=>?
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	@ABCDEFGHIJKLMNO
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	PQRSTUVWXYZ[\]^_
6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	`abcdefghijklmnopqrstuvwxyz
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	{ }~
8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
a	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿
b	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎ
c	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
d	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	àáâãäåæçèéêëìíî
e	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	ïðñóôõö÷øùúûüýþÿ
f	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	




total bytes : 256

shows that there are, indeed, one of each byte value. The total shown is a count of bytes.

By allowing python's decode() method to replace invalid UTF-8 bytes with the standard U_FFFD replacement character (flag -e):

```
utfa.py 256 -e

unicode analysis for 256 (running Python 3)




      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f -- unicode block --      -- characters --
0      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      .....
1      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      .....
2      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      !"#$%&'()*+,-./
3      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      0123456789:;<=>?
4      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      @ABCDEFGHIJKLMNO
5      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      PQRSTUVWXYZ[\]^_
6      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      `abcdefghijklmnopqrstuvwxyz
7      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 Basic Latin      {!}~
0xffff0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 128 0 0 Specials    

total bytes      : 256
total characters : 256
```

Note that the count spacing is adjusted to match the width of the largest character count (here 128). Various flags can be enabled to include or suppress parts of the output, and the table of individual counts can be replaced with summary totals for each block of sixteen consecutive characters (flag -x):

```
utfa.py 256 -ex

unicode analysis for 256 (running Python 3)

total -- unicode block --      -- characters --
0      16 Basic Latin      .....
1      16 Basic Latin      .....
2      16 Basic Latin      !"#$%&'()*+,-./
3      16 Basic Latin      0123456789:;<=>?
4      16 Basic Latin      @ABCDEFGHIJKLMNO
5      16 Basic Latin      PQRSTUVWXYZ[\]^_
6      16 Basic Latin      `abcdefghijklmnopqrstuvwxyz
7      16 Basic Latin      {!}~
0xffff0 128 Specials        

total bytes      : 256
total characters : 256
```

Depending on operating system character handling, normally most Unicode characters are displayed correctly, but a few special characters may be unprintable, reverse video, or have unusual vertical spacing or print directions (e.g. Hebrew, Arabic). The 'Specials' here show this. It is possible to display only the characters actually present in the file/web page rather than the full set of sixteen values (flag -m):

```
utfa.py 256 -exm

unicode analysis for 256 (running Python 3)
```

```

total -- unicode block --      -- characters --

0  16 Basic Latin              .....
1  16 Basic Latin              .....
2  16 Basic Latin              !"#$%&'()*+,-./
3  16 Basic Latin              0123456789;<=>?
4  16 Basic Latin              @ABCDEFGHIJKLMNO
5  16 Basic Latin              PQRSTUVWXYZ[\]^_
6  16 Basic Latin              `abcdefghijklmnopqrstuvwxyz
7  16 Basic Latin              {!}~
0xffff 128 Specials           

total bytes      : 256
total characters : 256

```

Here, flag -m masks all but the 0xffff in the last line, replacing all others with a space.

Since '256' is not valid UTF-8, here it has been converted to 'u256':

```
utf8.py u256
```

```
unicode analysis for u256 (running Python 3)
```

	0 1 2 3 4 5 6 7 8 9 a b c d e f	-- unicode block --	-- characters --
0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin
1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin
2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin	!"#\$%&'()*+,-./
3	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin	0123456789;<=>?
4	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin	@ABCDEFGHIJKLMNO
5	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin	PQRSTUVWXYZ[\]^_
6	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin	`abcdefghijklmnopqrstuvwxyz
7	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Basic Latin	{!}~
0x80	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement
0x90	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement
0xa0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement	¡¢£¥¦§¨ª«¬®¯
0xb0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement	°±²³´µ¶·¸¹º»¼½¿
0xc0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement	ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎ
0xd0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement	ÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
0xe0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement	àáâäåæçèéêëìíî
0xf0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Latin-1 Supplement	ñòóôõö÷øùúûýþÿ

```
total bytes      : 384
total characters : 256
```

Note that for characters above 0x7f rows are shown only where at least one example of a character in each group of sixteen is present in the input. Now the two totals at the end show bytes and characters. Each byte value above 0x7f in the original file requires two bytes in UTF-8. This can be seen by analysing 'u256' as bytes:

```
utf8.py u256 -b
```

```
byte analysis for u256
```

	0 1 2 3 4 5 6 7 8 9 a b c d e f	-- characters --
--	---------------------------------	------------------

[illegible]

```
total bytes : 384
```

Of course, although the legend is headed 'characters' the characters displayed here are not true UTF-8 characters, but those same glyphs are part of ISO/IEC 8859, so it is helpful to make the values distinct.

Example usage

Having shown '256' many different ways, more interesting inputs can show the program in other useful cases.

Here is the Wikipedia web page on UTF-8, referencing the web page using flag -u:

```
utf8.py -u http://en.wikipedia.org/wiki/UTF-8
```

```
unicode analysis for http://en.wikipedia.org/wiki/UTF-8      (running Python 3)
```

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	-- unicode block --	-- characters --
0	0	0	0	0	0	0	0	0	0	1827	3029	0	0	0	0	0	Basic Latin
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Basic Latin
2	15449	21	10545	495	6	801	761	60	486	486	33	384	441	2871	1686	9036	Basic Latin	!"#\$%&'()*+,-./
3	2679	2023	1731	1154	573	607	720	614	1430	776	1431	1614	11509	5926	11509	202	Basic Latin	0123456789:;<=>?
4	3	644	350	1335	328	573	1383	134	161	612	129	140	229	376	164	257	Basic Latin	@ABCDEFGHIJKLMNO
5	263	28	226	704	897	937	76	176	50	23	92	119	28	119	38	1941	Basic Latin	PQRSTUVWXYZ[\]^_
6	1	16224	3487	6586	6581	16639	3880	3266	4254	14765	118	2439	11454	3892	10620	8188	Basic Latin	`abcdefghijklmnopqrstuvwxyz
7	6585	110	8941	11721	13555	2391	1295	2824	1419	1983	236	36	1	36	6	0	Basic Latin	~{ }~
0xa0	0	0	2	1	0	0	0	9	0	0	0	0	0	0	1	0	Latin-1 Supplement	¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯
0xb0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	Latin-1 Supplement	° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾
0xc0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Latin-1 Supplement	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
0xd0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Latin-1 Supplement	Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
0xe0	1	0	0	0	0	2	0	2	0	2	1	0	0	0	0	1	Latin-1 Supplement	à á â ã ä å æ ç è é ê ë ì í î ï
0xf0	0	1	0	2	0	0	0	0	0	0	0	0	3	2	0	0	Latin-1 Supplement	ð ñ ò ó ô õ ö ù ú û ü ý þ ÿ
0x100	0	8	0	0	0	0	0	0	0	0	0	0	1	2	0	0	Latin Extended-A	Ā ā Ă ă Ą ą Å å Ć ć Č č Ĉ ĉ Ċ ċ
0x120	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	Latin Extended-A	Ĝ ĝ Ğ ğ Ĥ ĥ Ħ ħ İ i Ĳ ĳ Ĵ ĵ
0x150	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	Latin Extended-A	Ō ō Œ œ Š š ſ Š š Š š Š š Š š Š š

```
total bytes      : 269811
total characters : 269307
```

The standard Unicode Basic Multilingual Plane (BMP) of 65,536 characters (out of 65,536 possible values) is insufficient to include all Unicode characters. Python3 on OSX is compiled to support 32-bit Unicode characters internally, so can handle characters from the Supplementary Multilingual Plane (SMP) and Supplementary Ideographic Plane (SIP) correctly. However, Python2 supports only the Basic Multilingual Plane and converts any out-of-range unicode to pairs of 16-bit values (surrogate pairs). Running the same

version of the analysis program with the python 2.7 interpreter shows (the unchanged middle part of the output is removed here for brevity):

```
python2 utfa.py -u http://en.wikipedia.org/wiki/UTF-8

unicode analysis for http://en.wikipedia.org/wiki/UTF-8      (running Python 2 : showing Basic Multilingual Plane including surrogate pairs)

      0      1      2      3      4      5      6      7      8      9      a      b      c      d      e      f      -- unicode block --      -- characters --
      0      0      0      0      0      0      0      0      0      0 1827 3029      0      0      0      0      0      0      Basic Latin      .....
      1      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      Basic Latin      .....
...
0x20a0      0      0      0      0      0      0      0      0      0      0      0      0      0      4      0      0      0      Currency Symbols      €£¢¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿
0x4e20      0      0      0      0      0      0      0      0      0      0      0      0      0      0      1      0      0      0      CJK Unified Ideographs      北南丟舛𠂇𠂉𠂊𠂋𠂌𠂍𠂎𠂏𠂐𠂑𠂒𠂓𠂔𠂕𠂖𠂗𠂘𠂙𠂚𠂛𠂜𠂝𠂞𠂟𠂠𠂡𠂢𠂣𠂤𠂥𠂦𠂧𠂨𠂩𠂪𠂫𠂬𠂭𠂮𠂯𠂰𠂱𠂲𠂳𠂴𠂵𠂶𠂷𠂸𠂹𠂺𠂻𠂼𠂽𠂾𠂿𠃀𠃁𠃂𠃃𠃄𠃅𠃆𠃇𠃈𠃉𠃊𠃋𠃌𠃍𠃎𠃏𠃐𠃑𠃒𠃓𠃔𠃕𠃖𠃗𠃘𠃙𠃚𠃛𠃜𠃝𠃞𠃟𠃠𠃡𠃢𠃣𠃤𠃥𠃦𠃧𠃨𠃩𠃪𠃫𠃬𠃭𠃮𠃯𠃰𠃱𠃲𠃳𠃴𠃵𠃶𠃷𠃸𠃹𠃺𠃻𠃼𠃽𠃾𠃿𠄀𠄁𠄂𠄃𠄄𠄅𠄆𠄇𠄈𠄉𠄊𠄋𠄌𠄍𠄎𠄏𠄐𠄑𠄒𠄓𠄔𠄕𠄖𠄗𠄘𠄙𠄚𠄛𠄜𠄝𠄞𠄟𠄠𠄡𠄢𠄣𠄤𠄥𠄦𠄧𠄨𠄩𠄪𠄫𠄬𠄭𠄮𠄯𠄰𠄱𠄲𠄳𠄴𠄵𠄶𠄷𠄸𠄹𠄺𠄻𠄼𠄽𠄾𠄿𠅀𠅁𠅂𠅃𠅄𠅅𠅆𠅇𠅈𠅉𠅊𠅋𠅌𠅍𠅎𠅏𠅐𠅑𠅒𠅓𠅔𠅕𠅖𠅗𠅘𠅙𠅚𠅛𠅜𠅝𠅞𠅟𠅠𠅡𠅢𠅣𠅤𠅥𠅦𠅧𠅨𠅩𠅪𠅫𠅬𠅭𠅮𠅯𠅰𠅱𠅲𠅳𠅴𠅵𠅶𠅷𠅸𠅹𠅺𠅻𠅼𠅽𠅾𠅿𠆀𠆁𠆂𠆃𠆄𠆅𠆆𠆇𠆈𠆉𠆊𠆋𠆌𠆍𠆎𠆏𠆐𠆑𠆒𠆓𠆔𠆕𠆖𠆗𠆘𠆙𠆚𠆛𠆜𠆝𠆞𠆟𠆠𠆡𠆢𠆣𠆤𠆥𠆦𠆧𠆨𠆩𠆪𠆫𠆬𠆭𠆮𠆯𠆰𠆱𠆲𠆳𠆴𠆵𠆶𠆷𠆸𠆹𠆺𠆻𠆼𠆽𠆾𠆿𠇀𠇁𠇂𠇃𠇄𠇅𠇆𠇇𠇈𠇉𠇊𠇋𠇌𠇍𠇎𠇏𠇐𠇑𠇒𠇓𠇔𠇕𠇖𠇗𠇘𠇙𠇚𠇛𠇜𠇝𠇞𠇟𠇠𠇡𠇢𠇣𠇤𠇥𠇦𠇧𠇨𠇩𠇪𠇫𠇬𠇭𠇮𠇯𠇰𠇱𠇲𠇳𠇴𠇵𠇶𠇷𠇸𠇹𠇺𠇻𠇼𠇽𠇾𠇿𠈀𠈁𠈂𠈃𠈄𠈅𠈆𠈇𠈈𠈉𠈊𠈋𠈌𠈍𠈎𠈏𠈐𠈑𠈒𠈓𠈔𠈕𠈖𠈗𠈘𠈙𠈚𠈛𠈜𠈝𠈞𠈟𠈠𠈡𠈢𠈣𠈤𠈥𠈦𠈧𠈨𠈩𠈪𠈫𠈬𠈭𠈮𠈯𠈰𠈱𠈲𠈳𠈴𠈵𠈶𠈷𠈸𠈹𠈺𠈻𠈼𠈽𠈾𠈿𠉀𠉁𠉂𠉃𠉄𠉅𠉆𠉇𠉈𠉉𠉊𠉋𠉌𠉍𠉎𠉏𠉐𠉑𠉒𠉓𠉔𠉕𠉖𠉗𠉘𠉙𠉚𠉛𠉜𠉝𠉞𠉟𠉠𠉡𠉢𠉣𠉤𠉥𠉦𠉧𠉨𠉩𠉪𠉫𠉬𠉭𠉮𠉯𠉰𠉱𠉲𠉳𠉴𠉵𠉶𠉷𠉸𠉹𠉺𠉻𠉼𠉽𠉾𠉿𠊀𠊁𠊂𠊃𠊄𠊅𠊆𠊇𠊈𠊉𠊊𠊋𠊌𠊍𠊎𠊏𠊐𠊑𠊒𠊓𠊔𠊕𠊖𠊗𠊘𠊙𠊚𠊛𠊜𠊝𠊞𠊟𠊠𠊡𠊢𠊣𠊤𠊥𠊦𠊧𠊨𠊩𠊪𠊫𠊬𠊭𠊮𠊯𠊰𠊱𠊲𠊳𠊴𠊵𠊶𠊷𠊸𠊹𠊺𠊻𠊼𠊽𠊾𠊿𠋀𠋁𠋂𠋃𠋄𠋅𠋆𠋇𠋈𠋉𠋊𠋋𠋌𠋍𠋎𠋏𠋐𠋑𠋒𠋓𠋔𠋕𠋖𠋗𠋘𠋙𠋚𠋛𠋜𠋝𠋞𠋟𠋠𠋡𠋢𠋣𠋤𠋥𠋦𠋧𠋨𠋩𠋪𠋫𠋬𠋭𠋮𠋯𠋰𠋱𠋲𠋳𠋴𠋵𠋶𠋷𠋸𠋹𠋺𠋻𠋼𠋽𠋾𠋿𠌀𠌁𠌂𠌃𠌄𠌅𠌆𠌇𠌈𠌉𠌊𠌋𠌌𠌍𠌎𠌏𠌐𠌑𠌒𠌓𠌔𠌕𠌖𠌗𠌘𠌙𠌚𠌛𠌜𠌝𠌞𠌟𠌠𠌡𠌢𠌣𠌤𠌥𠌦𠌧𠌨𠌩𠌪𠌫𠌬𠌭𠌮𠌯𠌰𠌱𠌲𠌳𠌴𠌵𠌶𠌷𠌸𠌹𠌺𠌻𠌼𠌽𠌾𠌿𠍀𠍁𠍂𠍃𠍄𠍅𠍆𠍇𠍈𠍉𠍊𠍋𠍌𠍍𠍎𠍏𠍐𠍑𠍒𠍓𠍔𠍕𠍖𠍗𠍘𠍙𠍚𠍛𠍜𠍝𠍞𠍟𠍠𠍡𠍢𠍣𠍤𠍥𠍦𠍧𠍨𠍩𠍪𠍫𠍬𠍭𠍮𠍯𠍰𠍱𠍲𠍳𠍴𠍵𠍶𠍷𠍸𠍹𠍺𠍻𠍼𠍽𠍾𠍿𠎀𠎁𠎂𠎃𠎄𠎅𠎆𠎇𠎈𠎉𠎊𠎋𠎌𠎍𠎎𠎏𠎐𠎑𠎒𠎓𠎔𠎕𠎖𠎗𠎘𠎙𠎚𠎛𠎜𠎝𠎞𠎟𠎠𠎡𠎢𠎣𠎤𠎥𠎦𠎧𠎨𠎩𠎪𠎫𠎬𠎭𠎮𠎯𠎰𠎱𠎲𠎳𠎴𠎵𠎶𠎷𠎸𠎹𠎺𠎻𠎼𠎽𠎾𠎿𠏀𠏁𠏂𠏃𠏄𠏅𠏆𠏇𠏈𠏉𠏊𠏋𠏌𠏍𠏎𠏏𠏐𠏑𠏒𠏓𠏔𠏕𠏖𠏗𠏘𠏙𠏚𠏛𠏜𠏝𠏞𠏟𠏠𠏡𠏢𠏣𠏤𠏥𠏦𠏧𠏨𠏩𠏪𠏫𠏬𠏭𠏮𠏯𠏰𠏱𠏲𠏳𠏴𠏵𠏶𠏷𠏸𠏹𠏺𠏻𠏼𠏽𠏾𠏿�0�1�2�3�4�5�6�7�8�9�a�b�c𠐄𠐅𠐆𠐇𠐈𠐉𠐊𠐋𠐌𠐍𠐎𠐏𠐐𠐑𠐒𠐓𠐔𠐕𠐖𠐗𠐘𠐙𠐚𠐛𠐜𠐝𠐞𠐟𠐠𠐡𠐢𠐣𠐤𠐥𠐦𠐧𠐨𠐩𠐪𠐫𠐬𠐭𠐮𠐯𠐰𠐱𠐲𠐳𠐴𠐵𠐶𠐷𠐸𠐹𠐺𠐻𠐼𠐽𠐾𠐿�0�1�2�3�4�5�6�7�8�9�a�b�c𠑄𠑅�f𠑇𠑈𠑉𠑊𠑋𠑌𠑍𠑎𠑏𠑐𠑑𠑒𠑓𠑔𠑕𠑖𠑗𠑘𠑙𠑚𠑛𠑜𠑝𠑞𠑟𠑠𠑡𠑢𠑣𠑤𠑥𠑦𠑧𠑨𠑩𠑪𠑫𠑬𠑭𠑮𠑯𠑰𠑱𠑲𠑳𠑴𠑵𠑶𠑷𠑸𠑹𠑺𠑻𠑼𠑽𠑾𠑿𠒀𠒁𠒂𠒃𠒄𠒅𠒆𠒇𠒈𠒉𠒊𠒋𠒌𠒍𠒎𠒏𠒐𠒑𠒒𠒓𠒔𠒕𠒖𠒗𠒘𠒙𠒚𠒛𠒜𠒝𠒞𠒟𠒠𠒡𠒢𠒣𠒤𠒥𠒦𠒧𠒨𠒩𠒪𠒫𠒬𠒭𠒮𠒯𠒰𠒱𠒲𠒳𠒴𠒵𠒶𠒷𠒸𠒹𠒺𠒻𠒼𠒽𠒾𠒿�0�1�2�3�4�5�6�7�8�9�a𠓂𠓃𠓄𠓅𠓆𠓇𠓈𠓉𠓊𠓋𠓌𠓍𠓎𠓏𠓐𠓑𠓒𠓓𠓔𠓕𠓖𠓗𠓘𠓙𠓚𠓛𠓜𠓝𠓞𠓟𠓠𠓡𠓢𠓣𠓤𠓥𠓦𠓧𠓨𠓩𠓪𠓫𠓬𠓭𠓮𠓯𠓰𠓱𠓲𠓳𠓴𠓵𠓶𠓷𠓸𠓹𠓺𠓻𠓼𠓽𠓾𠓿�0�1�2�3�4�5�6�7�8�9�a𠔂𠔃𠔄𠔅𠔆𠔇𠔈𠔉𠔊𠔋𠔌𠔍𠔎𠔏𠔐𠔑𠔒𠔓𠔔𠔕𠔖𠔗𠔘𠔙𠔚𠔛𠔜𠔝𠔞𠔟𠔠𠔡𠔢𠔣𠔤𠔥𠔦𠔧𠔨𠔩𠔪𠔫𠔬𠔭𠔮𠔯𠔰𠔱𠔲𠔳𠔴𠔵𠔶𠔷𠔸𠔹𠔺𠔻𠔼𠔽𠔾𠔿�0�1�2�3�4�5�6�7�8�9�a𠕂𠕃𠕄𠕅𠕆𠕇𠕈𠕉𠕊𠕋𠕌𠕍𠕎𠕏𠕐𠕑𠕒𠕓𠕔𠕕𠕖𠕗𠕘𠕙𠕚𠕛𠕜𠕝𠕞𠕟𠕠𠕡𠕢𠕣𠕤𠕥𠕦𠕧𠕨𠕩𠕪𠕫𠕬𠕭𠕮𠕯𠕰𠕱𠕲𠕳𠕴𠕵𠕶𠕷𠕸𠕹𠕺𠕻𠕼𠕽𠕾𠕿�0�1�2�3�4�5�6�7�8�9�a𠖂𠖃𠖄𠖅𠖆𠖇𠖈𠖉𠖊𠖋𠖌𠖍𠖎𠖏𠖐𠖑𠖒𠖓𠖔𠖕𠖖𠖗𠖘𠖙𠖚𠖛𠖜𠖝𠖞𠖟𠖠𠖡𠖢𠖣𠖤𠖥𠖦𠖧𠖨𠖩𠖪𠖫𠖬𠖭𠖮𠖯𠖰𠖱𠖲𠖳𠖴𠖵𠖶𠖷𠖸𠖹𠖺𠖻𠖼𠖽𠖾𠖿�0�1�2�3�4�5�6�7�8�9�a𠗂𠗃𠗄𠗅𠗆𠗇𠗈𠗉𠗊𠗋𠗌𠗍𠗎𠗏𠗐𠗑𠗒𠗓𠗔𠗕𠗖𠗗𠗘𠗙𠗚𠗛𠗜𠗝𠗞𠗟𠗠𠗡𠗢𠗣𠗤𠗥𠗦𠗧𠗨𠗩𠗪𠗫𠗬𠗭𠗮𠗯𠗰𠗱𠗲𠗳𠗴𠗵𠗶𠗷𠗸𠗹𠗺𠗻𠗼𠗽𠗾𠗿�0�1�2�3�4�5�6�7�8�9�a𠘂𠘃𠘄𠘅𠘆𠘇𠘈𠘉𠘊𠘋𠘌𠘍𠘎𠘏𠘐𠘑𠘒𠘓𠘔𠘕𠘖𠘗𠘘𠘙𠘚𠘛𠘜𠘝𠘞𠘟𠘠𠘡𠘢𠘣𠘤𠘥𠘦𠘧𠘨𠘩𠘪𠘫𠘬𠘭𠘮𠘯𠘰𠘱𠘲𠘳𠘴𠘵𠘶𠘷𠘸𠘹𠘺𠘻𠘼𠘽𠘾𠘿�0�1�2�3�4�5�6�7�8�9�a𠙂𠙃𠙄𠙅𠙆𠙇𠙈𠙉𠙊𠙋𠙌𠙍𠙎𠙏𠙐𠙑𠙒𠙓𠙔𠙕𠙖𠙗𠙘𠙙𠙚𠙛𠙜𠙝𠙞𠙟𠙠𠙡𠙢𠙣𠙤𠙥𠙦𠙧𠙨𠙩𠙪𠙫𠙬𠙭𠙮𠙯𠙰𠙱𠙲𠙳𠙴𠙵𠙶𠙷𠙸𠙹𠙺𠙻𠙼𠙽𠙾𠙿�0�1�2�3�4�5�6�7�8�9�a𠚂𠚃𠚄𠚅𠚆𠚇𠚈𠚉𠚊𠚋𠚌𠚍𠚎𠚏𠚐𠚑𠚒𠚓𠚔𠚕𠚖𠚗𠚘𠚙𠚚𠚛𠚜𠚝𠚞𠚟𠚠𠚡𠚢𠚣𠚤𠚥𠚦𠚧𠚨𠚩𠚪𠚫𠚬𠚭𠚮𠚯𠚰𠚱𠚲𠚳𠚴𠚵𠚶𠚷𠚸𠚹𠚺𠚻𠚼𠚽𠚾𠚿�0�1�2�3�4�5�6�7�8�9�a𠛂𠛃𠛄𠛅𠛆𠛇𠛈𠛉𠛊𠛋𠛌𠛍𠛎𠛏𠛐𠛑𠛒𠛓𠛔𠛕𠛖𠛗𠛘𠛙𠛚𠛛𠛜𠛝𠛞𠛟𠛠𠛡𠛢𠛣𠛤𠛥𠛦𠛧𠛨𠛩𠛪𠛫𠛬𠛭𠛮𠛯𠛰𠛱𠛲𠛳𠛴𠛵𠛶𠛷𠛸𠛹𠛺𠛻𠛼𠛽𠛾𠛿�0�1�2�3�4�5�6�7�8�9�a𠜂𠜃𠜄𠜅𠜆𠜇𠜈𠜉𠜊𠜋𠜌𠜍𠜎𠜏𠜐𠜑𠜒𠜓𠜔𠜕𠜖𠜗𠜘𠜙𠜚𠜛𠜜𠜝𠜞𠜟𠜠𠜡𠜢𠜣𠜤𠜥𠜦𠜧𠜨𠜩𠜪𠜫𠜬𠜭𠜮𠜯𠜰𠜱𠜲𠜳𠜴𠜵𠜶𠜷𠜸𠜹𠜺𠜻𠜼𠜽𠜾𠜿�0�1�2�3�4�5�6�7�8�9�a𠝂𠝃𠝄𠝅𠝆𠝇𠝈𠝉𠝊𠝋𠝌𠝍𠝎𠝏𠝐𠝑𠝒𠝓𠝔𠝕𠝖𠝗𠝘𠝙𠝚𠝛𠝜𠝝𠝞𠝟𠝠𠝡𠝢𠝣𠝤𠝥𠝦𠝧𠝨𠝩𠝪𠝫𠝬𠝭𠝮𠝯𠝰𠝱𠝲𠝳𠝴𠝵𠝶𠝷𠝸𠝹𠝺𠝻𠝼𠝽𠝾𠝿𠞀𠞁𠞂𠞃𠞄𠞅𠞆𠞇𠞈𠞉𠞊𠞋𠞌𠞍𠞎𠞏𠞐𠞑𠞒𠞓𠞔𠞕𠞖𠞗𠞘𠞙𠞚𠞛𠞜𠞝𠞞𠞟𠞠𠞡𠞢𠞣𠞤𠞥𠞦𠞧𠞨𠞩𠞪𠞫𠞬𠞭𠞮𠞯𠞰𠞱𠞲𠞳𠞴𠞵𠞶𠞷𠞸𠞹𠞺𠞻𠞼𠞽𠞾𠞿𠟀𠟁𠟂𠟃𠟄𠟅𠟆𠟇𠟈𠟉𠟊𠟋𠟌𠟍𠟎𠟏𠟐𠟑𠟒𠟓𠟔𠟕𠟖𠟗𠟘𠟙𠟚𠟛𠟜𠟝𠟞𠟟𠟠𠟡𠟢𠟣𠟤𠟥𠟦𠟧𠟨𠟩𠟪𠟫𠟬𠟭𠟮𠟯𠟰𠟱𠟲𠟳𠟴𠟵𠟶𠟷𠟸𠟹𠟺𠟻𠟼𠟽𠟾𠟿�0�1�2�3�4�5�6�7�8�9�a𠠂𠠃𠠄𠠅𠠆𠠇𠠈𠠉𠠊𠠋𠠌𠠍𠠎𠠏𠠐𠠑𠠒𠠓𠠔𠠕𠠖𠠗𠠘𠠙𠠚𠠛𠠜𠠝𠠞𠠟𠠠𠠡𠠢𠠣𠠤𠠥𠠦𠠧𠠨𠠩𠠪𠠫𠠬𠠭𠠮𠠯𠠰𠠱𠠲𠠳𠠴𠠵𠠶𠠷𠠸𠠹𠠺𠠻𠠼𠠽𠠾𠠿�0�1�2�3�4�5�6�7�8�9�a𠡂𠡃𠡄𠡅𠡆𠡇𠡈𠡉𠡊𠡋𠡌𠡍𠡎𠡏𠡐𠡑𠡒𠡓𠡔𠡕𠡖𠡗𠡘𠡙𠡚𠡛𠡜𠡝𠡞𠡟𠡠𠡡𠡢𠡣𠡤𠡥𠡦𠡧𠡨𠡩𠡪𠡫𠡬𠡭𠡮𠡯𠡰𠡱𠡲𠡳𠡴𠡵𠡶𠡷𠡸𠡹𠡺𠡻𠡼𠡽𠡾𠡿�0�1�2�3�4�5�6�7�8�9�a𠢂𠢃𠢄𠢅𠢆𠢇𠢈𠢉𠢊𠢋𠢌𠢍𠢎𠢏𠢐𠢑𠢒𠢓𠢔𠢕𠢖𠢗𠢘𠢙𠢚𠢛𠢜𠢝𠢞𠢟𠢠𠢡𠢢𠢣𠢤𠢥𠢦𠢧𠢨𠢩𠢪𠢫𠢬𠢭𠢮𠢯𠢰𠢱𠢲𠢳𠢴𠢵𠢶𠢷𠢸𠢹𠢺𠢻𠢼𠢽𠢾𠢿�0�1�2�3�4�5�6�7�8�9�a𠣂𠣃𠣄𠣅𠣆𠣇𠣈𠣉𠣊𠣋𠣌𠣍𠣎𠣏𠣐𠣑𠣒𠣓𠣔𠣕𠣖𠣗𠣘𠣙𠣚𠣛𠣜𠣝𠣞𠣟𠣠𠣡𠣢𠣣𠣤𠣥𠣦𠣧𠣨𠣩𠣪𠣫𠣬𠣭𠣮𠣯𠣰𠣱𠣲𠣳𠣴𠣵𠣶𠣷𠣸𠣹𠣺𠣻𠣼𠣽𠣾𠣿�0�1�2�3�4�5�6�7�8�9�a𠤂𠤃𠤄𠤅𠤆𠤇𠤈𠤉𠤊𠤋𠤌𠤍𠤎𠤏𠤐𠤑𠤒𠤓𠤔𠤕𠤖𠤗𠤘𠤙𠤚𠤛𠤜𠤝𠤞𠤟𠤠𠤡𠤢𠤣𠤤𠤥𠤦𠤧𠤨𠤩𠤪𠤫𠤬𠤭𠤮𠤯𠤰𠤱𠤲𠤳𠤴𠤵𠤶𠤷𠤸𠤹𠤺𠤻𠤼𠤽𠤾𠤿𠥀𠥁𠥂𠥃𠥄𠥅𠥆𠥇𠥈𠥉𠥊𠥋𠥌𠥍𠥎𠥏𠥐𠥑𠥒𠥓𠥔𠥕𠥖𠥗𠥘𠥙𠥚𠥛𠥜𠥝𠥞𠥟𠥠𠥡𠥢𠥣𠥤𠥥𠥦𠥧𠥨𠥩𠥪𠥫𠥬𠥭𠥮𠥯𠥰𠥱𠥲𠥳𠥴𠥵𠥶𠥷𠥸𠥹𠥺𠥻𠥼𠥽𠥾𠥿𠦀𠦁𠦂𠦃𠦄𠦅𠦆𠦇𠦈𠦉𠦊𠦋𠦌𠦍𠦎𠦏𠦐𠦑𠦒𠦓𠦔𠦕𠦖𠦗𠦘𠦙𠦚𠦛𠦜𠦝𠦞𠦟𠦠𠦡𠦢𠦣𠦤𠦥𠦦𠦧𠦨𠦩𠦪𠦫𠦬𠦭𠦮𠦯𠦰𠦱𠦲𠦳𠦴𠦵𠦶𠦷𠦸𠦹𠦺𠦻𠦼𠦽𠦾𠦿𠧀𠧁𠧂𠧃𠧄𠧅𠧆𠧇𠧈𠧉𠧊𠧋𠧌𠧍𠧎𠧏𠧐𠧑𠧒𠧓𠧔𠧕𠧖𠧗𠧘𠧙𠧚𠧛𠧜𠧝𠧞𠧟𠧠𠧡𠧢𠧣𠧤𠧥𠧦𠧧𠧨𠧩𠧪𠧫𠧬𠧭𠧮𠧯𠧰𠧱𠧲𠧳𠧴𠧵𠧶𠧷𠧸𠧹𠧺𠧻𠧼𠧽𠧾𠧿𠨀𠨁𠨂𠨃𠨄𠨅𠨆𠨇𠨈𠨉𠨊𠨋𠨌𠨍𠨎𠨏𠨐𠨑𠨒𠨓𠨔𠨕𠨖𠨗𠨘𠨙𠨚𠨛𠨜𠨝𠨞𠨟𠨠𠨡𠨢𠨣𠨤𠨥𠨦𠨧𠨨𠨩𠨪𠨫𠨬𠨭𠨮𠨯𠨰𠨱𠨲𠨳𠨴𠨵𠨶𠨷𠨸𠨹𠨺𠨻𠨼𠨽𠨾𠨿𠩀𠩁𠩂𠩃𠩄𠩅𠩆𠩇𠩈𠩉𠩊𠩋𠩌𠩍𠩎𠩏𠩐𠩑𠩒𠩓𠩔𠩕𠩖𠩗𠩘𠩙𠩚𠩛𠩜𠩝𠩞𠩟𠩠𠩡𠩢𠩣𠩤𠩥𠩦𠩧𠩨𠩩𠩪𠩫𠩬𠩭𠩮𠩯𠩰𠩱𠩲𠩳𠩴𠩵𠩶𠩷𠩸𠩹𠩺𠩻𠩼𠩽𠩾𠩿𠪀𠪁𠪂𠪃𠪄𠪅𠪆𠪇𠪈𠪉𠪊𠪋𠪌𠪍𠪎𠪏𠪐𠪑𠪒𠪓𠪔𠪕𠪖𠪗𠪘𠪙𠪚𠪛𠪜𠪝𠪞𠪟𠪠𠪡𠪢𠪣𠪤𠪥𠪦𠪧𠪨𠪩𠪪𠪫𠪬𠪭𠪮𠪯𠪰𠪱𠪲𠪳𠪴𠪵𠪶𠪷𠪸𠪹𠪺𠪻𠪼𠪽𠪾𠪿𠫀𠫁𠫂𠫃𠫄𠫅𠫆𠫇𠫈𠫉𠫊𠫋𠫌𠫍𠫎𠫏𠫐𠫑𠫒𠫓𠫔𠫕𠫖𠫗𠫘𠫙𠫚𠫛𠫜𠫝𠫞𠫟𠫠𠫡𠫢𠫣𠫤𠫥𠫦𠫧𠫨𠫩𠫪𠫫𠫬𠫭𠫮𠫯𠫰𠫱𠫲𠫳𠫴𠫵𠫶𠫷𠫸𠫹𠫺𠫻𠫼𠫽𠫾𠫿𠬀𠬁𠬂𠬃𠬄𠬅𠬆𠬇𠬈𠬉𠬊𠬋𠬌𠬍𠬎𠬏𠬐𠬑𠬒𠬓𠬔𠬕𠬖𠬗𠬘𠬙𠬚𠬛𠬜𠬝𠬞𠬟𠬠𠬡𠬢𠬣𠬤𠬥𠬦𠬧𠬨𠬩𠬪𠬫𠬬𠬭𠬮𠬯𠬰𠬱𠬲𠬳𠬴𠬵𠬶𠬷𠬸𠬹𠬺𠬻𠬼𠬽𠬾𠬿𠭀𠭁𠭂𠭃𠭄𠭅𠭆𠭇𠭈𠭉𠭊𠭋𠭌𠭍𠭎𠭏𠭐𠭑𠭒𠭓𠭔𠭕𠭖𠭗𠭘𠭙𠭚𠭛𠭜𠭝𠭞𠭟𠭠𠭡𠭢𠭣𠭤𠭥𠭦𠭧𠭨𠭩𠭪𠭫𠭬𠭭𠭮𠭯𠭰𠭱𠭲𠭳𠭴𠭵𠭶𠭷𠭸𠭹𠭺𠭻𠭼𠭽𠭾𠭿𠮀𠮁𠮂𠮃𠮄𠮅𠮆
```

(running Python 3)

[illegible]

[illegible]

Here there are three Braille characters (for 'b', 'r' and 'l'). They are a little hard to discover on the web page but can be found by expanding 'Type of writing systems' at the foot of the page. <brl> is a frequently-used braille contraction of the name 'braille' itself.

```
utf8.py -u http://bbc.co.uk -xm
```


unicode analysis for http://bbc.co.uk (running Python 3)

total -- unicode block --			-- characters --
0	241	Basic Latin	.
1	0	Basic Latin	
2	25365	Basic Latin	!"#\$%&'()*+,-./
3	11273	Basic Latin	0123456789:;<=>?
4	1767	Basic Latin	ABCDEFGHIJKLMNO
5	1605	Basic Latin	PQRSTUVWXYZ[\]^_
6	41106	Basic Latin	abcdefghijklmnopqrstuvwxyz
7	24356	Basic Latin	{}~
0xa0	2	Latin-1 Supplement	£
0x2010	4	General Punctuation	'
0x2020	3	General Punctuation	...

total bytes : 105738
total characters : 105722

shows that almost all characters are 'Basic Latin' except for a few pound signs and some 'General punctuation' marks.

There are clearly both new line and carriage return characters, but not in equal numbers (241 is odd). The full breakdown:

utf8.py -u http://bbc.co.uk

unicode analysis for http://bbc.co.uk (running Python 3)

																	-- unicode block --	-- characters --
0	0	0	0	0	0	0	0	0	0	0	179	0	0	62	0	0	Basic Latin
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Basic Latin
2	10491	170	2131	6	8	20	2434	872	771	771	169	172	435	1187	2372	3356	Basic Latin	!"#\$%&'()*+,-./
3	519	487	554	486	431	271	220	271	226	289	811	2711	1090	1737	1096	74	Basic Latin	0123456789:;<=>?
4	0	208	211	285	128	221	73	47	48	105	17	40	73	130	105	76	Basic Latin	@ABCDEFGHIJKLMNO
5	99	2	76	147	184	81	40	48	4	25	0	198	97	198	15	391	Basic Latin	PQRSTUVWXYZ[\]^_
6	0	4383	2035	3573	2479	5323	1179	1867	1911	4466	198	839	3359	1371	3197	4926	Basic Latin	`abcdefghijklmnopqrstuvwxyz
7	1969	892	3443	3907	7180	2294	987	1606	281	733	187	390	95	392	0	0	Basic Latin	{}~
0xa0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	Latin-1 Supplement	¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿
0x2010	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	General Punctuation	—…‰‘’‚“”„
0x2020	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	General Punctuation	†‡•….

total bytes : 105738
total characters : 105722

shows many more new line characters (179) than carriage returns (62). This is a little unusual because most DOS-format files contain these in pairs.

Command line options

Here are the command line controls for utfa.py:

```
utfa.py -h
usage: utfa [-h] [-u URL] [-b] [-e] [-t] [-l] [-n] [-m] [-x] [-o OFFSET]
           [-s SIZE] [-v]
           [file]
```

file analysis

positional arguments:
file

optional arguments:

-h, --help	show this help message and exit
-u URL, --url URL	url e.g. -u http://bbc.co.uk
-b, --bytes	analyse 256 possible bit patterns
-e, --errors	enable unicode error replacement
-t, --time	show run time
-l, --legend	disable legend display
-n, --name	disable unicode block names
-m, --mask	show only characters present in file/url
-x, --number	disable individual counts
-o OFFSET, --offset OFFSET	
-s SIZE, --size SIZE	
-v, --version	python version, encoding and max code point