

Text File Coding

Leonardo Araújo

UFSJ



Introduction to Text Encoding

- **Why Encoding Matters:**

- Facilitates communication between computers and humans.
- Ensures data integrity across different systems.

- **Overview:**

- Evolution from simple to complex encoding systems.

Morse Code

- **History:** Developed by Samuel F.B. Morse in the early 1840s.
- **Mechanism:** Uses dots, dashes, and spaces for letters, numbers, and punctuation.
- **Usage:** Primarily telegraphy, but also in radio communication.

A	• -	N	- •	1	• - - - -
B	- • • •	O	- - - -	2	• • - - -
C	- • • • •	P	• - - •	3	• • • - -
D	- • •	Q	- - • • -	4	• • • • -
E	•	R	• • • •	5	• • • • •
F	• • • •	S	• • •	6	- • • • •
G	- • • •	T	-	7	- - • • •
H	• • • •	U	• • •	8	- - • • • •
I	• •	V	• • • -	9	- - • • • • •
J	• • - - -	W	• • - -	0	- - • • • • -
K	- • • -	X	- • • • -	.	• • • • • •
L	• • • •	Y	- • • • -	,	- - • • • • -
M	- -	Z	- • • • •	?	• • - • • •

Figure 1: Morse Code

Jacquard

- Joseph Marie Jacquard in Lyon in 1801.

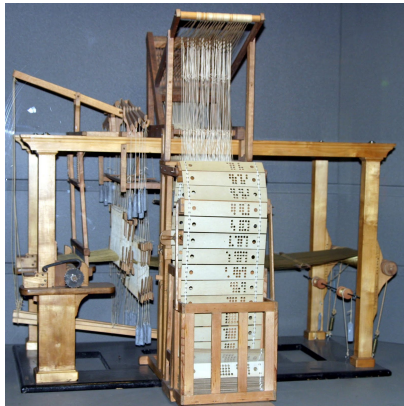


Figure 2: Jacquard's loom.

Baudot and Murray Code

■ Baudot Code:

- Invented by Émile Baudot, 5-bit code for telegraphy.
- Limited characters, used shift for numbers/letters.

■ Murray Code (ITA2):

- Extension of Baudot, improved by Donald Murray.
- Added lower case, more symbols.

LETTERS FIGURES		A	B	C	D WHO ARE YOU	E	F	G	H	I	J BELL	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	CARRIAGE RETURN	LINE FEED	LETTERS	FIGURES	SPACE	ALL-SPACE NOT IN USE
CODE ELEMENTS	1	●	●		●	●					●	●						●		●		●		●	●	●	●			●	●		
	2	● ○		●			●	●		●	●	●	●				●	●	●			●	●	●	●	●			●	●	●		
	3		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	4		●	●	●		●	●	●		●	●		●	●	●	●		●			●	●		●	●		●		●	●		
	5		●					●	●			●	●	●		●	●	●			●		●	●	●	●	●	●			●	●	

● INDICATES A MARK ELEMENT (A HOLE PUNCHED IN THE TAPE)

○ INDICATES POSITION OF A SPROCKET HOLE IN THE TAPE

The International Telegraph Alphabet

Figure 3: ITA2 Baudot-Murray code

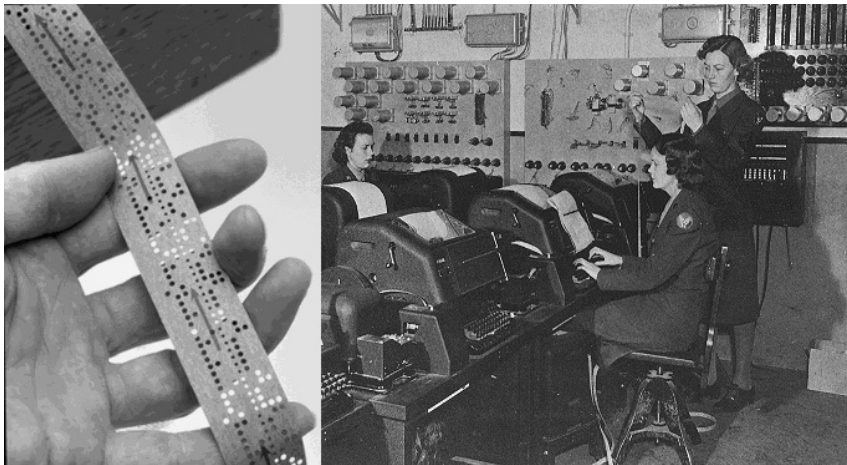
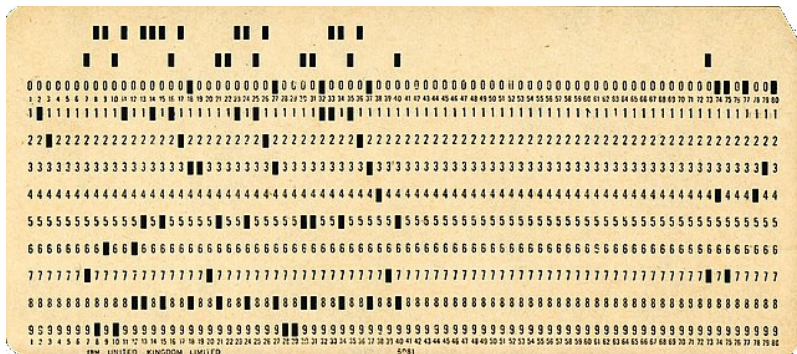


Figure 4: Teletype and perforated strip.

EBCDIC (Extended Binary Coded Decimal Interchange Code)

- **History:** Developed by IBM for mainframe computers.
- **Characteristics:** Used in legacy systems.
- EBCD, a subset of EBCDIC.



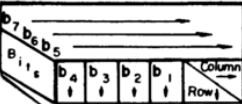
	-	1	2	3	4	5	6	7	8	9	2-8	3-8	4-8	5-8	6-8	7-8
-		1	2	3	4	5	6	7	8	9	:	#	@	'	=	"
Y	&	A	B	C	D	E	F	G	H	I	[.	<	(+	!
X	-	J	K	L	M	N	O	P	Q	R]	\$	*)	:	^
0	0	/	S	T	U	V	W	X	Y	Z	\	,	%	_	>	?

Figure 5: 12-row/80-column IBM punched card and EBCD table.

ASCII and Extended ASCII

- **ASCII (American Standard Code for Information Interchange):**
 - 7-bit code, 128 characters including 33 non-printing control codes.
 - Standardized in 1963 (ANSI).
 - Backward Compatibility: Despite its age, ASCII remains widely used today for compatibility reasons.
- **Extended ASCII:**
 - 8-bit code, 256 characters, allowing for additional symbols and characters.

USASCII code chart



Column Row	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	@	P	\	p	
1	SOH	DC1	!	A	Q	a	q	
2	STX	DC2	"	B	R	b	r	
3	ETX	DC3	#	C	S	c	s	
4	EOT	DC4	\$	D	T	d	t	
5	ENQ	NAK	%	E	U	e	u	
6	ACK	SYN	&	F	V	f	v	
7	BEL	ETB	'	G	W	g	w	
8	BS	CAN	(H	X	h	x	
9	HT	EM)	I	Y	i	y	
10	LF	SUB	*	J	Z	j	z	
11	VT	ESC	+	K	[k	{	
12	FF	FS	,	L	\	l		
13	CR	GS	-	M]	m	}	
14	SO	RS	.	N	^	n	~	
15	SI	US	/	O	_	o	DEL	

Figure 6: ASCII Table

Character	Binary (Uppercase)	Binary (Lowercase)	Character
A	01000001	01100001	a
B	01000010	01100010	b
C	01000011	01100011	c
...
Z	01011010	01111010	z
1	00110001	00100001	!
2	00110010	01000000	"
3	00110011	00100011	#
4	00110100	00100100	\$
...

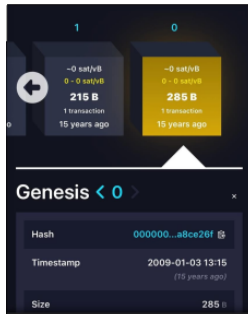
ASCII Art

```

| .-----|
||                                     || | | | | | | |
||           -----                 ||
||           . ; ; ; ; ; ; ; .      ||
||           / ; ; ; ; ; ; ; ; \    ||
||           / ; / ^ ^ ^ ^ ^ ^ ^ ^ . . ||
||           | ; | _ _ _ _ \ ; ; ; | ||
|| | . . | ; | e ^ / e ^ | ; ; ; | ||
||           | ; | | | | | ; ; ; | ' -- ||
||           | ; | | ^ _ | ; ; ; | ||
||           | ; \ -- ^ / | ; ; ; | ||
||           | ; ; ; ; ; --- ^ \ | ; ; ; | ||
||           | ; ; ; ; | | | ; ; ; | ||
||           | ; ; . - ^ | ; ; ; | ||
|| ' -- | / ^ | ; ; ; | -- . ||
|| ; ; ; ; . ; ; ; ; \ ; ; ; | ||
|| ; ; ; ; - . ; _ / . - ; ; ; ; | ||
|| ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; | ||
|-----|

```

BTC Genesis Block



Bitcoin Genesis Block

Raw Hex Version

```

00000000 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000020 00 00 00 00 3B A3 ED FD 7A 7B 12 B2 7A C7 2C 3E .....;life{.*q>
00000030 67 76 8F 61 7F C8 1B C3 88 8A 51 32 3A 9F 88 AA gv.a.B.A`S02;V,*
00000040 4B 1E 5E 4A 29 AB 5F 49 FF FF 00 1D 1D AC 2B 7C K."j)*_Iyy...~+|
00000050 01 01 00 00 01 00 00 00 00 00 00 00 00 00 00 00 .....
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 00 00 00 00 00 FF FF FF FF 4D 04 FF FF 00 1D .....yyyym.yy..
00000080 01 04 45 54 68 65 20 54 69 6D 65 73 20 30 33 2F ...The Times 03/
00000090 4A 61 6E 2F 32 30 30 39 20 43 68 61 6E 63 65 6C lor on brink of
000000A0 6C 6F 72 20 6F 6E 20 62 72 69 6E 6B 20 6F 66 20 second bailout f
000000B0 73 65 63 6F 6E 64 20 62 61 69 6C 6F 75 74 20 66 or bankspppyf..b.
000000C0 6F 72 20 62 61 6E 6B 73 FF FF FF FF 01 00 F2 05 *....CA.g5p"psh"
000000D0 2A 01 00 00 00 43 41 04 67 8A FD D0 FE 55 48 27 .gn/q0*.\0' (A9.
000000E0 19 67 F1 A6 71 30 37 10 5C DE A8 28 D0 39 09 A6 yb&e.ap*10k7L18K
000000F0 79 62 E0 EA 1F 61 DE B6 49 F6 BC 3F 4C EF 38 C4 dU.A.A.b\BN+*.W
00000100 F3 55 04 K5 1E C1 12 DE 5C 38 4D P7 BA 0B 8D 57 $!p+kh._'....
00000110 8A 4C 70 2B 6B F1 1D 5F AC 00 00 00 00

```



Figure 7: The message embedded by Satoshi Nakamoto in Bitcoin's first block (the Genesis Block). The message reads, "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks," which was a headline from The Times newspaper on that date.

Base64

Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format. Each Base64 digit represents exactly 6 bits of data, providing a way to encode binary data as text.

Base64 is used in:

- email attachments,
- embedding binary data in XML, JSON, or HTML, and
- data exchange in APIs.

```
$ base64 /tmp/tux.png
iVBORw0KGgoAAAANSUHEUgAAADIAAAA7CAYAAAA5MNl5AAAAxHpUWHRSYXcgCHJvZmlsZSB0eXB1
IGV4aWYAAHjabVBbDsMgDPvnFDsCiVMix6GPSbvBjr8AaVW2WcINceSahOP9eoZHA5MEWbKmkLI0
SJHC1QqNA7UzRencAXaN5n7A6gJbC21yXDX5/Nmny2B8qLXLzUg3F9ZZKOL++mXkidAstXp3o7Jd
kbtAbldHs2Iqmu9PWI84Q8cJjUTn2D/3bNvbF/sPmA8QojGgIwDakYBqAhszkg0SxGpB7p3TzBby
b08nwgcmkllHdJ9h5QAAAYRpQ0NQSUNDIHByb2ZpbGUAAHicfZE9SMNAHMVfU0WRiogdpDhkqLrY
RUUcaxWkUCHUCq06mFz6BU0akhQXR8G140DHYtXBxVlXB1dBEPwAcXZwUnsREv+XFFrEeHDcj3f3
HnfvAKFRYZrVFQc03TbTyYSYza2KPa8IIYJBjCMoM8uYk6QUfMfXPQJ8vYvxLP9zf45+NW8xICAS
x5lh2sQbxDObtsF5nzjMSrJKfE48YdIFiR+5rnj8xrnossAzW2YmPU8cJhaLHax0MCuZGvE0cVTV
dMoXsh6rnLc4a5Uaa92TvzCU1leWuU5zBEksYgkSRCiooYwKbMRo1UmxxKb9hI8/4volcinkKoOR
YwFVaJBdP/gf/07WkKxNekmhBND94jgfo0DPLtCs0873seM0T4DgM3ClT/3VBjD7SXq9rUWPgIFt
40K6rSl7wOUOMPxkyKbsSkGaQqEAvJ/RN+WAoVugb83rrbWP0wgcQ12lboCDQ2CsSNnrPu/u7eZt
3z0t/n4AkJJZysiZoe0AAA14aVRYdFhNTDpj20uYWRvYmUueG1wAAAAAAA8P3hwYWNrZXQgYmVn
aw49Iu+7vyIgaWQ9Iic1TTBNCENlaGlIenJlU3p0VGNg6a2M5ZCI/Pgo8eDp4bXBtZXRhIHhtbG5z
Ong9ImFkb2JlOm5zOm1ldGEvIiB40nhtcHRrPSJYTVAgQ29yZSA0LjQuMC1FeGl2MiI+CIA8cmRm
OLJERiB4bWxuc2pyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvdMDIvMjItcmRmLXN5bnRheC1u
```



Figure 8: Encoding of Tux image into base64.

ASCII Smuggling

ASCII smuggling is a technique that leverages Unicode characters, which are invisible in user interfaces but can be interpreted by large language models (LLMs), to embed hidden instructions or data within text. This method allows attackers to manipulate AI responses or exfiltrate sensitive information without the user's awareness, by embedding these hidden Unicode tags within clickable hyperlinks or documents shared in chats.

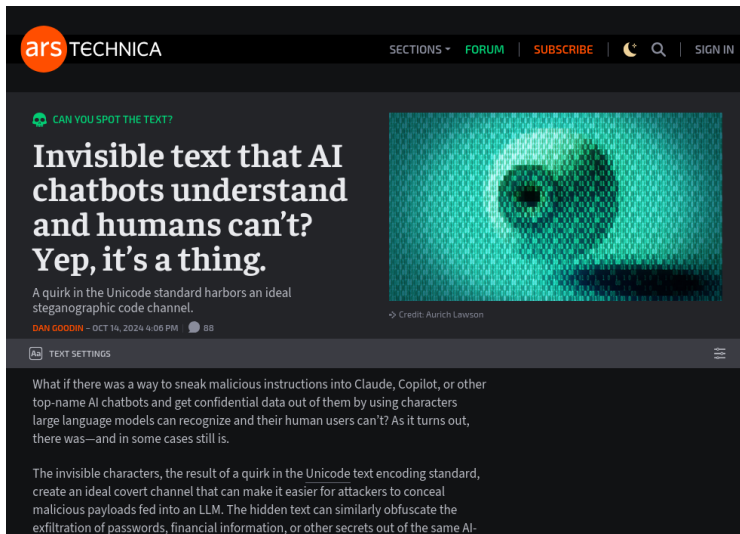


Figure 9: Ars Technica

ASCII Smuggler

Convert ASCII text to Unicode Tags which are invisible in most UI elements.

Check if a text has hidden Unicode Tags embedded with Decode.

Can you spot the text?
Invisible text that AI chatbots understand and humans can't? Yep, it's a thing.
A quirk in the Unicode standard harbors an ideal steganographic code channel.

Encode

Decode

[Advanced Options](#)

Can you spot the **Easter Egg** text?
Invisible text that AI chatbots understand and humans can't? Yep, it's a thing.
A quirk in the Unicode standard harbors an ideal steganographic code channel.

Hidden Unicode Tags discovered.

Clear

Figure 10: <https://embracethered.com/blog/ascii-smuggler.html>

ISO/IEC Standards

■ ISO/IEC 8859:

- Series for 8-bit character encoding supporting multiple languages.
- ISO-8859-1 (Western Europe), also known as ISO Latin 1.
 - The first 128 characters are identical to ASCII.
 - 0x00 to 1F and 0x80 to 0x9F (hex) used for C0 and C1 control codes.
 - C0 set was originally defined in ISO 646 (ASCII) (e.g., Start of Heading, Start of Text, End of Text, End of Transmission, ...).
 - C1 are additional control codes (e.g., Padding Character, High Octet Preset, Break Permitted Here, No Break Here, ...).

■ ISO/IEC 10646:

- Universal character set (UCS) for multilingual text.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	MUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	:	;	<	=	>	?
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	} 007D	~ 007E	DEL 007F
80																
90																
A0	NBSP 00A0	ı 00A1	ç 00A2	£ 00A3	¤ 00A4	¥ 00A5	ı 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	– 00AD	® 00AE	— 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C0	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D0	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E0	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F0	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	÷ 00F6	ø 00F7	ù 00F8	ú 00F9	û 00FA	ü 00FB	ý 00FC	ÿ 00FD		

Figure 11: ISO-8859-1 code page

Windows Code Pages

- **Overview:**

- Multiple code pages for different regions and languages.

- **Examples:**

- CP1252 (Western Europe), CP932 (Japan)

- **Issues:**

- Inconsistencies across different systems.

Unicode

■ Unicode:

- Universal character set covering all scripts, supporting over 143,000 characters.
- It assigns a unique number (called a “code point”) to each character, regardless of platform, program, or language.
- 1.112.064 valid code points within the codespace.
- As of Unicode 16.0, released in September 2024, 299,056 (27%) of these code points are allocated, 155,063 (14%) have been assigned characters, 137,468 (12%) are reserved for private use, 2,048 are used to enable the mechanism of surrogates, and 66 are designated as noncharacters, leaving the remaining 815,056 (73%) unallocated.
- Unicode has different encoding forms: UTF-8, UTF-16, and UTF-32.

UTF

- **UTF-8:**
 - Variable-length encoding, backward compatible with ASCII, byte-order independent.
- **UTF-16:**
 - Variable-length encoding (2 or 4 bytes per character).
 - Latin and most commonly used CJK¹ characters are encoded in 2 bytes.
- **UTF-32:**
 - Fixed-length encoding (4 bytes per character).

¹Chinese, Japanese, and Korean.

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
1	U+0000	U+007F	0xxxxxxx					
2	U+0080	U+07FF	110xxxxx	10xxxxxx				
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
4	U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
5	U+200000	U+3FFFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
6	U+4000000	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Figure 12: UTF-8 Structure

UTF-8 takes over

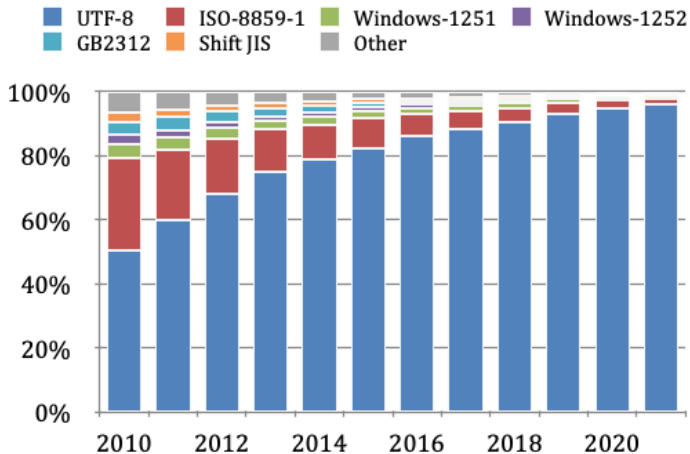


Figure 13: Declared character set for the 10 million most popular websites since 2010

Endianness

- **Big Endian vs. Little Endian:**
 - Byte order in memory representation.
 - Impacts how multi-byte characters are read.



Figure 14: Endianness

■ Example:

- UTF-16 and UTF-32 can be big or little endian.
- Byte Order Mark (BOM) to indicate the endianness being used.
- The BOM is the code point U+FEFF (BOM, ZWNBSP²).
 - Big-endian (UTF-16BE): FE FF
 - Little-endian (UTF-16LE): FF FE
 - Big-endian (UTF-32BE): 00 00 FE FF
 - Little-endian (UTF-32LE): FF FE 00 00
 - BOM in UTF-8: EF BB BF, serves more as a signature to indicate that the file is encoded in UTF-8 rather than specifying byte order.

²zero width no-break space

Text File Formats

- **.txt:** Usually ASCII or UTF-8.
- **.csv:** Can use various encodings; important for data exchange.
- **.json:** UTF-8 by default, supports Unicode.
- **.log:** Log files for recording events, errors, and system activities.

Markup Files

- **TeX:**
 - **Usage:** Typesetting language for high-quality typography.
 - **Encoding:** Often UTF-8, but can be sensitive to non-ASCII characters without proper preamble setup.
- **XML (eXtensible Markup Language):**
 - **Purpose:** Used for structured data storage and transmission.
 - **Encoding:** Declared in XML declaration, typically UTF-8 or UTF-16. Encoding declaration is crucial for correct parsing.
- **HTML (HyperText Markup Language):**
 - **Usage:** Standard markup language for documents designed to be displayed in a web browser.
 - **Encoding:** Default is often UTF-8, but can be specified with the `charset` attribute in the `<meta>` tag. Incorrect encoding can lead to garbled text.
- **Markdown:**
 - **Purpose:** Lightweight markup language for formatting text.
 - **Encoding:** Generally UTF-8, supports rich text with simple syntax.

Linux Tools for Text Encoding

- **iconv:** Converts text from one encoding to another.
 - Example: `iconv -f ISO-8859-1 -t UTF-8 input.txt > output.txt`
- **file:** Identifies file types and encodings.
 - Example: `file -i example.txt`
- **uconv:** More advanced conversion with Unicode support.
 - Example: `uconv -f UTF-8 -t UTF-16 input.txt -o output.txt`
- **dos2unix / unix2dos:** Converts between Windows and Unix line endings.
 - Example: `dos2unix file.txt` (converts CRLF to LF)
 - Example: `unix2dos file.txt` (converts LF to CRLF)
- **Use Cases:**
 - Data migration, cleaning, and internationalization.

Conclusion

- **Key Points:**

- Text encoding has evolved from simple to complex systems.
- Unicode provides a universal solution for global text representation.

- **Future:**

- Continued development in encoding standards to support new scripts and symbols.