

# Codificação de Arquivos de Texto

Leonardo Araújo

UFSJ



# Introdução à Codificação de Texto

## ■ Por que a codificação é importante:

- Formato legível para humanos: A codificação de texto permite que computadores armazenem e troquem textos de forma legível para humanos e processável por máquinas.
- Padronização: Codificações como UTF-8 fornecem um modo padrão para interpretar texto, evitando confusões causadas por codificações regionais ou proprietárias.
- Garantia de integridade dos dados entre sistemas diferentes.
  - A codificação adequada preserva a precisão do texto ao compartilhá-lo, prevenindo corrupção de dados e garantindo a exibição consistente de caracteres.

## ■ Visão Geral:

- Evolução de sistemas de codificação simples para complexos.

# Introdução à Codificação de Texto

## ■ Por que a codificação é importante:

- Formato legível para humanos: A codificação de texto permite que computadores armazenem e troquem textos de forma legível para humanos e processável por máquinas.
- Padronização: Codificações como UTF-8 fornecem um modo padrão para interpretar texto, evitando confusões causadas por codificações regionais ou proprietárias.
- Garantia de integridade dos dados entre sistemas diferentes.
  - A codificação adequada preserva a precisão do texto ao compartilhá-lo, prevenindo corrupção de dados e garantindo a exibição consistente de caracteres.

## ■ Visão Geral:

- Evolução de sistemas de codificação simples para complexos.

# Introdução à Codificação de Texto

## ■ Por que a codificação é importante:

- Formato legível para humanos: A codificação de texto permite que computadores armazenem e troquem textos de forma legível para humanos e processável por máquinas.
- Padronização: Codificações como UTF-8 fornecem um modo padrão para interpretar texto, evitando confusões causadas por codificações regionais ou proprietárias.
- Garantia de integridade dos dados entre sistemas diferentes.
  - A codificação adequada preserva a precisão do texto ao compartilhá-lo, prevenindo corrupção de dados e garantindo a exibição consistente de caracteres.

## ■ Visão Geral:

- Evolução de sistemas de codificação simples para complexos.

# Introdução à Codificação de Texto

## ■ Por que a codificação é importante:

- Formato legível para humanos: A codificação de texto permite que computadores armazenem e troquem textos de forma legível para humanos e processável por máquinas.
- Padronização: Codificações como UTF-8 fornecem um modo padrão para interpretar texto, evitando confusões causadas por codificações regionais ou proprietárias.
- Garantia de integridade dos dados entre sistemas diferentes.
  - A codificação adequada preserva a precisão do texto ao compartilhá-lo, prevenindo corrupção de dados e garantindo a exibição consistente de caracteres.

## ■ Visão Geral:

- Evolução de sistemas de codificação simples para complexos.

# Introdução à Codificação de Texto

## ■ Por que a codificação é importante:

- Formato legível para humanos: A codificação de texto permite que computadores armazenem e troquem textos de forma legível para humanos e processável por máquinas.
- Padronização: Codificações como UTF-8 fornecem um modo padrão para interpretar texto, evitando confusões causadas por codificações regionais ou proprietárias.
- Garantia de integridade dos dados entre sistemas diferentes.
  - A codificação adequada preserva a precisão do texto ao compartilhá-lo, prevenindo corrupção de dados e garantindo a exibição consistente de caracteres.

## ■ Visão Geral:

- Evolução de sistemas de codificação simples para complexos.

## Caractere vs. Glifo vs. Fonte

- **Caractere:** Unidade abstrata na codificação (ex.: 'A' no Unicode).
- **Glifo:** Forma visual de um caractere (como 'A' aparece em Arial vs. Times).
- **Fonte:** Coleção de glifos que compartilham um estilo de design.



Figura 1: Glifo vs fonte.

# Jacquard

- Joseph Marie Jacquard em Lyon, 1801.

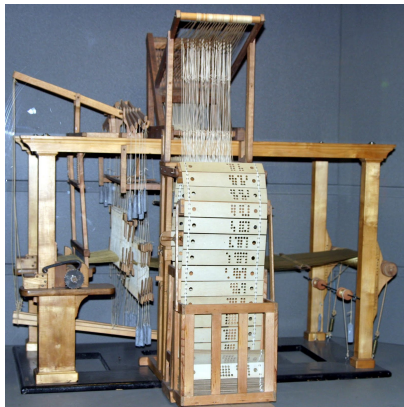


Figura 2: Jacquard's loom.



## Escrita Noturna e Braille: Evolução na Codificação Tátil

### ■ Escrita Noturna:

- Inventor: Charles Barbier, 1815, para comunicação militar silenciosa.
- Estrutura: Células de 12 pontos para sons fonéticos, complexa para uso prático.

### ■ Braille:

- Criador: Louis Braille, 1824 (primeira publicação em 1829), adaptado da Escrita Noturna.
- Inovação: Célula de 6 pontos, mais simples e acessível para deficientes visuais.
- Codificação Binária: Cada célula representa combinações binárias.
- Adoção Universal: Braille tornou-se o padrão mundial de comunicação para deficientes visuais.
- Expansão: Adaptado para várias línguas, matemática, música e outros.

## Escrita Noturna e Braille: Evolução na Codificação Tátil

### ■ Escrita Noturna:

- Inventor: Charles Barbier, 1815, para comunicação militar silenciosa.
- Estrutura: Células de 12 pontos para sons fonéticos, complexa para uso prático.

### ■ Braille:

- Criador: Louis Braille, 1824 (primeira publicação em 1829), adaptado da Escrita Noturna.
- Inovação: Célula de 6 pontos, mais simples e acessível para deficientes visuais.
- Codificação Binária: Cada célula representa combinações binárias.
- Adoção Universal: Braille tornou-se o padrão mundial de comunicação para deficientes visuais.
- Expansão: Adaptado para várias línguas, matemática, música e outros.

a	i	o	u	é	è
an	in	on	un	eu	ou
b	d	g	j	v	z
p	t	q	ch	f	s

	1	2	3	4	5	6
1	a	i	o	u	é	è
2	an	in	on	un	eu	ou
3	b	d	g	j	v	z
4	p	t	q	ch	f	s
5	l	m	n	r	gn	ll
6	oi	oin	ian	ien	ion	ieu

LOUIS BRAILLE'S ORIGINAL FRENCH ALPHABET

A	B	C	D	E	F	G	H	I	J
K	L	M	N	O	P	Q	R	S	T
U	V	X	Y	Z	ç	é	à	ê	û
an	in	on	un	eu	ou	oi	ch	gn	will
,	:	:	.	?	!	(	"	*	×

Figura 3: Escrita Noturna e Braille.

## Código Morse

- **História:** Desenvolvido por Samuel Morse e Alfred Vail nos anos 1840.
- **Mecanismo:** Usa pontos, traços e espaços para letras, números e pontuações.
- **Uso:** Principalmente telegrafia, mas também comunicação por rádio.

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 -·-·-·	? ·-·-·-·

Figura 4: Morse Code.

# Código Baudot e Código Murray

## ■ Código Baudot:

- Inventado por Émile Baudot, código de 5 bits para telegrafia.
- Caracteres limitados, usava *shift* para números/letras.

## ■ Código Murray (ITA2):

- Extensão do Baudot, melhorado por Donald Murray.
- Adicionou letras minúsculas e mais símbolos.

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 RETUN	LINE FEED	LETTERS	FIGURES	SPACE	ALL SPACE NOT IN USE
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**

Figura 5: Código ITA2 Baudot-Murray.

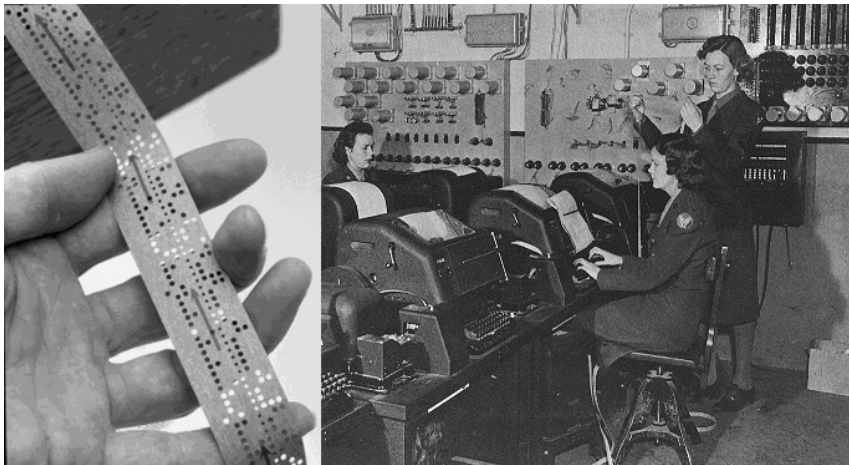
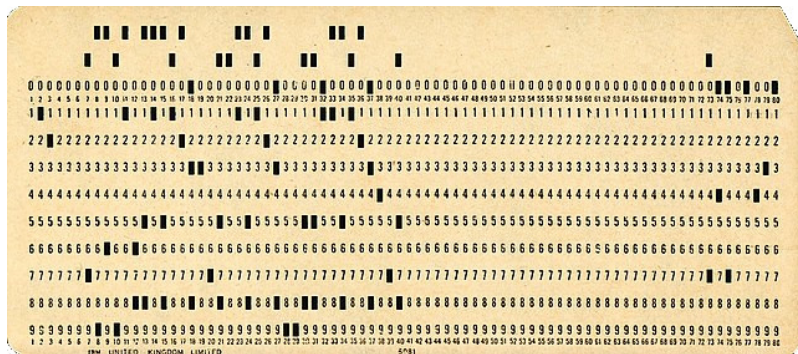


Figura 6: Teletipo e fita perfurada.

## EBCDIC (Extended Binary Coded Decimal Interchange Code)

- **História:** Desenvolvido pela IBM para computadores de grande porte.
- **Características:** Um dos primeiros sistemas de codificação de caracteres criados para processamento de dados em sistemas de larga escala.
  - Usado em sistemas legados (IBM 1401, 7090, System/360).
- EBCD, um subconjunto do 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	\	,	%	_	>	?

Figura 7: Cartão perfurado de 12 linhas/80 colunas e tabela EBDC.



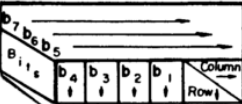
## ASCII e ASCII Estendido

- **ASCII (American Standard Code for Information Interchange):**
  - Código de 7 bits, 128 caracteres incluindo 33 códigos de controle não imprimíveis.
  - Padronizado em 1963 (ANSI).
  - Retrocompatibilidade: Apesar da idade, o ASCII continua amplamente utilizado hoje para garantir compatibilidade.
- **ASCII Estendido:**
  - Código de 8 bits, 256 caracteres, permitindo símbolos e caracteres adicionais.

## ASCII e ASCII Estendido

- **ASCII (American Standard Code for Information Interchange):**
  - Código de 7 bits, 128 caracteres incluindo 33 códigos de controle não imprimíveis.
  - Padronizado em 1963 (ANSI).
  - Retrocompatibilidade: Apesar da idade, o ASCII continua amplamente utilizado hoje para garantir compatibilidade.
- **ASCII Estendido:**
  - Código de 8 bits, 256 caracteres, permitindo símbolos e caracteres adicionais.

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	

Figura 8: Tabela ASCII.

Caractere	Binário (Maiúscula)	Binário (Minúscula)	Caractere
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	\$
...	...	...	...

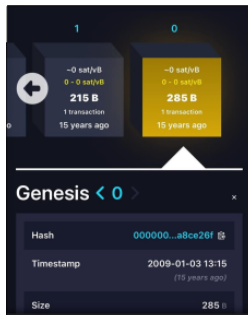
## ASCII Art

```

| .-----|
||                                     || | | | | |
||           -----                 ||
||           . ; ; ; ; ; ; ; .      ||
||           / ; ; ; ; ; ; ; ; \    ||
||           / ; / ^      ^ - ; ; ; ; . . ||
||           | ; | _ _      \ ; ; ; |   ||
|| | . - . | ; | 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 .....;fig{..*g,>
00000030 67 76 8F 61 7F C8 1B C3 88 8A 51 32 3A 9F B8 AA .....gv.a.B.A"8Q2iV,*
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 .....yyyyy.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 .....Jan/2009 Chancel
000000A0 6C 6F 72 20 6F 6E 20 62 72 69 6E 6B 20 6F 66 20 .....lor on brink of
000000B0 73 65 63 6F 6A 20 62 61 69 6C 6F 75 74 20 66 .....second bailout f
000000C0 6F 72 20 62 61 6E 6B 73 FF FF FF FF 01 00 F2 05 .....or banksffff..b.
000000D0 2A 01 00 00 00 43 41 04 67 8A FD DD FE 55 48 27 .....*....CA.g5p"pSH"
000000E0 19 67 F1 A6 71 30 37 10 5C DE A8 28 D0 39 09 A6 ....._gn/q0-.\"0\"(A9.;
000000F0 79 62 E0 EA 1F 61 DE B6 49 F6 BC 3F 4C EF 38 C4 .....ybb&.a?i0&7L&K&
00000100 F3 55 04 K5 1E C1 12 DE 5C 38 4D P7 BA 0B 8D 57 .....dU.A.A.b\"BN+*.W
00000110 8A 4C 70 2B 6B F1 1D 5F AC 00 00 00 00 .....$!p+kh._'....
  
```



Figura 9: A mensagem embutida por Satoshi Nakamoto no primeiro bloco do Bitcoin (o Bloco Gênesis). A mensagem diz: “The Times 03/Jan/2009 Chancellor on brink of second bailout for banks,” que era uma manchete do jornal The Times naquela data.

Mensagens ocultas em transações de Bitcoin são frequentemente inseridas usando o *opcode* OP\_RETURN, que permite armazenar até 80 bytes de dados (normalmente texto em ASCII) na saída da transação. Esse método é comumente usado para fins não financeiros, como inserir pequenos textos ou prova de dados.

# Base64

Base64 é um esquema de codificação de binário para texto que representa dados binários em um formato de string ASCII. Cada dígito Base64 representa exatamente 6 bits de dados, oferecendo uma forma de codificar dados binários como texto.

- Conjunto de caracteres: A B C D E F G H I J K L M N O P Q R S T U V W X Y  
Z a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 + /

Base64 é utilizado em:

- anexos de email,
- embutir dados binários em XML, JSON, ou HTML, e
- troca de dados em APIs.



# Base64

Base64 é um esquema de codificação de binário para texto que representa dados binários em um formato de string ASCII. Cada dígito Base64 representa exatamente 6 bits de dados, oferecendo uma forma de codificar dados binários como texto.

- Conjunto de caracteres: A B C D E F G H I J K L M N O P Q R S T U V W X Y  
Z a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 + /

Base64 é utilizado em:

- anexos de email,
- embutir dados binários em XML, JSON, ou HTML, e
- troca de dados em APIs.

Base64 alphabet defined in RFC 4648.

Index	Binary	Char.	Index	Binary	Char.	Index	Binary	Char.	Index	Binary	Char.
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
									Padding		=

Figura 10: Alfabeto Base64 definido na RFC 4648.

Quando o comprimento da entrada não codificada não for um múltiplo de três, a saída codificada deve ter preenchimento adicionado para que seu comprimento seja um múltiplo de quatro.

Input		Output		Padding
Text	Length	Text	Length	
<i>light work.</i>	11	bGlnaHQgd29yay4=	16	1
<i>light work</i>	10	bGlnaHQgd29yaw==	16	2
<i>light wor</i>	9	bGlnaHQgd29y	12	0
<i>light wo</i>	8	bGlnaHQgd28=	12	1
<i>light w</i>	7	bGlnaHQgdw==	12	2

Encoded	Padding	Length	Decoded
bGlnaHQgdw==	==	1	<i>light w</i>
bGlnaHQgd28=	=	2	<i>light wo</i>
bGlnaHQgd29y	None	3	<i>light wor</i>

Figura 11: *Padding* no Base64.

```
light w len=7 bGlnaHQgdw==  
w (ascii) 0111 0111  
011101 110000 000000  
d w
```

```
light wo len=8 bGlnaHQgd28=  
w (ascii) 0111 0111  
o (ascii) 0110 1111  
011101 110110 111100  
d 2 8
```

```
light wor len=9 bGlnaHQgd29y  
w (ascii) 0111 0111  
o (ascii) 0110 1111  
r (ascii) 0111 0010  
011101 110110 111101 110010  
d 2 9 y
```

```
$ base64 /tmp/tux.png
iVBORw0KGgoAAAANSUHEUgAAADIAAAA7CAYAAAA5MNl5AAAAxHpUWHRSYXcgCHJvZmlsZSB0eXB1
IGV4aWYAAHjabVBbDsMgDPvnFDsCiVMiX6GPSbvBjr8AaVW2WcINceSahOP9eoZHA5MEWbKmkLI0
SJHC1QqNA7UzRencAXaN5n7A6gJbC21yXDX5/Nmny2B8qLXLzUg3F9ZZKOL++mXkidAstXp3o7Jd
kbtAbldHs2Iqmu9PWI84Q8cJjUTn2D/3bNvbF/sPmA8QojGgIwDakYBqAhszkg0SxGpB7p3TzBby
b08nwgcmkllHdJ9h5QAAAYRpQ0NQSUNDIHByb2ZpbGUAAHicfZE9SMNAHMFU0WRiogdpDhkqLrY
RUUcaxWkUCHUCq06mFz6BU0akhQXR8G140DHYtXBxVlXB1dBEPwAcXZwUnsREv+XFFrEeHDcj3f3
HnfvAKFRYZrVFQc03TbTyYSYza2KPa8IIYJBjCMoM8uYk6QUfMfXPQJ8vYvxLP9zf45+NW8xICAS
x5lh2sQbxDObtsF5nzjMSrJKfE48YdIFiR+5rnj8xrnossAzW2YmPU8cJhaLHax0MCuZGvE0cVTV
dMoXsh6rnLc4a5Uaa92TvzCU1leWuU5zBEksYgkSRCiooYwKbMRo1UmxxKb9hI8/4volcinkKoOR
YwFVaJBdP/gf/07WkKxNekmhBND94jgfo0DPLtCs0873seM0T4DgM3ClT/3VBjD7SXq9rUWPgIFt
40K6rSl7wOUOMPxkyKbsSkGaQqEAvJ/RN+WAoVugb83rrbWP0wcgQ12lboCDQ2CsSNnrPu/u7ezt
3z0t/n4AkJJysiZoe0AAA14aVRYdFhNTDpj20uYWRvYmUueG1wAAAAAAA8P3hwYWNrZXQgYmVn
aw49Iu+7vyIgaWQ9Iic1TTBNcENlaGlIenJlU3p0VGNg6a2M5ZCI/Pgo8eDp4bXBtZXRhIHhtbG5z
Ong9ImFkb2JlOm5zOm1ldGEvIiB40nhtcHRrPSJYTVAgQ29yZSA0LjQuMC1FeGl2MiI+Cia8cmRm
OLJERiB4bWxuc2pyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvdMDIvMjItcmRmLXN5bnRheC1u
```



Figura 12: Codificação da imagem do Tux em base64.

## Codificação de Chaves e Endereços no Bitcoin

- Todas as chaves e endereços são codificados usando métodos apropriados:
  - **Base58Check**: Para endereços legados e chaves privadas.
  - **Bech32**: Para endereços SegWit.

### Prefix Summary Table

Tipo de Dado	Prefixo	Exemplo
Endereço Legado	0x00	1PMycacnJa...UAs
Endereço SegWit	bc1	bc1qw508d6q...
Endereço Testnet	0x6F	mhPo5P2RVu5...rEo
Chave Privada (WIF)	0x80	5J76fRXQYWk...U6q

## Base58Check

- Conjunto de caracteres: 1 2 3 4 5 6 7 8 9 A B C D E F G H J K L M N P Q R S T U V W X Y Z a b c d e f g h i j k m n o p q r s t u v w x y z.
- a-z, A-Z, e 0-9, com os caracteres visivelmente ambíguos (0, O, l, I) removidos.

## exemplo

- 3 bytes: 0xFFFFFFFF
- Base 58: 2UzHL
- Passos:
  - $0xFFFFFFFF = 16\,777\,215$
  - $16\,777\,215 \bmod 58 = 19 = L$
  - $289\,262 \bmod 58 = 16 = H$
  - $4987 \bmod 58 = 57 = z$
  - $85 \bmod 58 = 27 = U$
  - $1 \bmod 58 = 1 = 2$

## Base58Check

- Conjunto de caracteres: 1 2 3 4 5 6 7 8 9 A B C D E F G H J K L M N P Q R S T U V W X Y Z a b c d e f g h i j k m n o p q r s t u v w x y z.
- a-z, A-Z, e 0-9, com os caracteres visivelmente ambíguos (0, O, l, I) removidos.

## exemplo

- 3 bytes: 0xFFFFFFFF
- Base 58: 2UzHL
- Passos:
  - $0xFFFFFFFF = 16\,777\,215$
  - $16\,777\,215 \bmod 58 = 19 = L$
  - $289\,262 \bmod 58 = 16 = H$
  - $4987 \bmod 58 = 57 = z$
  - $85 \bmod 58 = 27 = U$
  - $1 \bmod 58 = 1 = 2$



## Bech32

- Conjunto de caracteres: q p z r y 9 x 8 g f 2 t v d w 0 s 3 j n 5 4 k h c e 6 m u a 7 l.
  - a-z, e 0-9, sem os seguintes caracteres: 1, b, i, o (b, i, o podem facilmente serem confundidos com 8, 1, 0, especialmente quando escritos à mão ou utilizando certas fontes).
  - Caracteres geralmente confundidos (e.g. 5 vs S, 2 vs Z, p vs q vs g, etc.) possuem sempre uma diferença de um bit – o código BCH é otimizado para detecção e correção de erros de um único bit.
  - código BCH, GF(32), polinômio
 
$$g(x) = x^6 + 29x^5 + 22x^4 + 20x^3 + 21x^2 + 29x + 18.$$
  - Detecção de erros: até 4 erros em 89 caracteres.
    - P2WPKH (Pay to Witness Public Key Hash): Esses endereços começam com bc1q e geralmente possuem 42 caracteres no mainnet (incluindo o prefixo bc1).
    - P2WSH (Pay to Witness Script Hash): Também começam com bc1q, mas são mais longos, tipicamente com 62 caracteres no mainnet, devido ao script hash ser maior.

## Bech32

- Conjunto de caracteres: q p z r y 9 x 8 g f 2 t v d w 0 s 3 j n 5 4 k h c e 6 m u a 7 l.
  - a-z, e 0-9, sem os seguintes caracteres: 1, b, i, o (b, i, o podem facilmente serem confundidos com 8, 1, 0, especialmente quando escritos à mão ou utilizando certas fontes).
  - Caracteres geralmente confundidos (e.g. 5 vs S, 2 vs Z, p vs q vs g, etc.) possuem sempre uma diferença de um bit – o código BCH é otimizado para detecção e correção de erros de um único bit.
  - código BCH, GF(32), polinômio
 
$$g(x) = x^6 + 29x^5 + 22x^4 + 20x^3 + 21x^2 + 29x + 18.$$
  - Detecção de erros: até 4 erros em 89 caracteres.
    - P2WPKH (Pay to Witness Public Key Hash): Esses endereços começam com bc1q e geralmente possuem 42 caracteres no mainnet (incluindo o prefixo bc1).
    - P2WSH (Pay to Witness Script Hash): Também começam com bc1q, mas são mais longos, tipicamente com 62 caracteres no mainnet, devido ao script hash ser maior.

## Bech32

- Conjunto de caracteres: q p z r y 9 x 8 g f 2 t v d w 0 s 3 j n 5 4 k h c e 6 m u a 7 l.
  - a-z, e 0-9, sem os seguintes caracteres: 1, b, i, o (b, i, o podem facilmente serem confundidos com 8, 1, 0, especialmente quando escritos à mão ou utilizando certas fontes).
  - Caracteres geralmente confundidos (e.g. 5 vs S, 2 vs Z, p vs q vs g, etc.) possuem sempre uma diferença de um bit – o código BCH é otimizado para detecção e correção de erros de um único bit.
  - código BCH, GF(32), polinômio
 
$$g(x) = x^6 + 29x^5 + 22x^4 + 20x^3 + 21x^2 + 29x + 18.$$
  - Detecção de erros: até 4 erros em 89 caracteres.
    - P2WPKH (Pay to Witness Public Key Hash): Esses endereços começam com bc1q e geralmente possuem 42 caracteres no mainnet (incluindo o prefixo bc1).
    - P2WSH (Pay to Witness Script Hash): Também começam com bc1q, mas são mais longos, tipicamente com 62 caracteres no mainnet, devido ao script hash ser maior.

## Bech32

- Conjunto de caracteres: q p z r y 9 x 8 g f 2 t v d w 0 s 3 j n 5 4 k h c e 6 m u a 7 l.
  - a-z, e 0-9, sem os seguintes caracteres: 1, b, i, o (b, i, o podem facilmente serem confundidos com 8, 1, 0, especialmente quando escritos à mão ou utilizando certas fontes).
  - Caracteres geralmente confundidos (e.g. 5 vs S, 2 vs Z, p vs q vs g, etc.) possuem sempre uma diferença de um bit – o código BCH é otimizado para detecção e correção de erros de um único bit.
  - código BCH, GF(32), polinômio
 
$$g(x) = x^6 + 29x^5 + 22x^4 + 20x^3 + 21x^2 + 29x + 18.$$
  - Detecção de erros: até 4 erros em 89 caracteres.
    - P2WPKH (Pay to Witness Public Key Hash): Esses endereços começam com bc1q e geralmente possuem 42 caracteres no mainnet (incluindo o prefixo bc1).
    - P2WSH (Pay to Witness Script Hash): Também começam com bc1q, mas são mais longos, tipicamente com 62 caracteres no mainnet, devido ao script hash ser maior.

## ASCII Smuggling

O ASCII *smuggling* é uma técnica que utiliza caracteres Unicode de uma região específica conhecida como *Tags Unicode Block*. Esses caracteres são invisíveis nas interfaces de usuário, mas podem ser interpretados por modelos de linguagem grandes (LLMs). Ao converter caracteres ASCII em equivalentes de marca Unicode, os atacantes podem embutir instruções ou dados ocultos em um texto aparentemente inofensivo, fazendo parecer que não há informações adicionais, enquanto essas informações ainda são acionáveis por sistemas de IA. Esse método permite que atacantes manipulem respostas de IA, exfiltrem informações sensíveis ou alterem links clicáveis ou documentos, tudo sem o conhecimento do usuário.

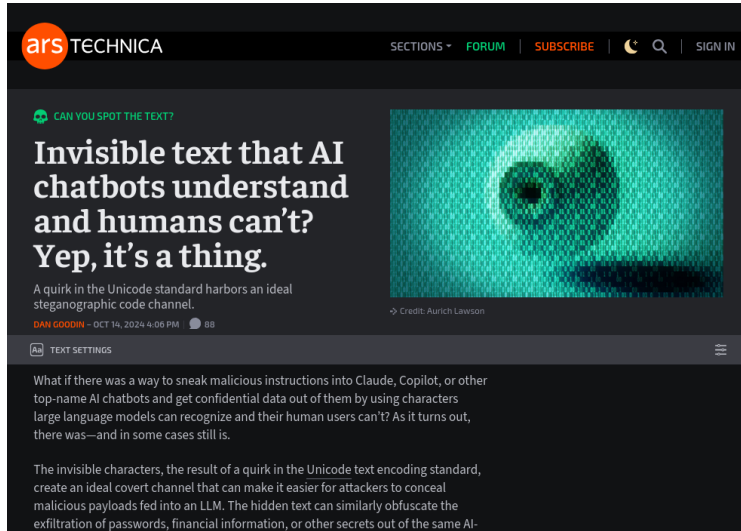


Figura 13: Ars Technica.

## 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

Figura 14: <https://embracethered.com/blog/ascii-smuggler.html>

## Padrões ISO/IEC

### ■ ISO/IEC 8859:

- Série de codificação de caracteres de 8 bits que suporta múltiplos idiomas.
- ISO-8859-1 (Europa Ocidental), também conhecido como ISO Latin 1.
  - Os primeiros 128 caracteres são idênticos ao ASCII.
  - De 0x00 a 1F e de 0x80 a 0x9F (hex) usados para códigos de controle C0 e C1.
  - O conjunto C0 foi originalmente definido no ISO 646 (ASCII) (ex.: Início de Cabeçalho, Início de Texto, Fim de Texto, Fim de Transmissão, ...).
  - C1 são códigos de controle adicionais (ex.: Caractere de Preenchimento, Pré-ajuste de Alto Octeto, Quebra Permitida Aqui, Sem Quebra Aqui, ...).

### ■ ISO/IEC 10646:

- Conjunto universal de caracteres (UCS) para texto multilíngue.



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>MUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	<u>!</u> 0021	<u>"</u> 0022	<u>#</u> 0023	<u>\$</u> 0024	<u>%</u> 0025	<u>&amp;</u> 0026	<u>'</u> 0027	<u>(</u> 0028	<u>)</u> 0029	<u>*</u> 002A	<u>+</u> 002B	<u>,</u> 002C	<u>-</u> 002D	<u>.</u> 002E	<u>/</u> 002F
30	<u>0</u> 0030	<u>1</u> 0031	<u>2</u> 0032	<u>3</u> 0033	<u>4</u> 0034	<u>5</u> 0035	<u>6</u> 0036	<u>7</u> 0037	<u>8</u> 0038	<u>9</u> 0039	<u>:</u> 003A	<u>;</u> 003B	<u>&lt;</u> 003C	<u>=</u> 003D	<u>&gt;</u> 003E	<u>?</u> 003F
40	<u>@</u> 0040	<u>A</u> 0041	<u>B</u> 0042	<u>C</u> 0043	<u>D</u> 0044	<u>E</u> 0045	<u>F</u> 0046	<u>G</u> 0047	<u>H</u> 0048	<u>I</u> 0049	<u>J</u> 004A	<u>K</u> 004B	<u>L</u> 004C	<u>M</u> 004D	<u>N</u> 004E	<u>O</u> 004F
50	<u>P</u> 0050	<u>Q</u> 0051	<u>R</u> 0052	<u>S</u> 0053	<u>T</u> 0054	<u>U</u> 0055	<u>V</u> 0056	<u>W</u> 0057	<u>X</u> 0058	<u>Y</u> 0059	<u>Z</u> 005A	<u>[</u> 005B	<u>\</u> 005C	<u>]</u> 005D	<u>^</u> 005E	<u>_</u> 005F
60	<u>`</u> 0060	<u>a</u> 0061	<u>b</u> 0062	<u>c</u> 0063	<u>d</u> 0064	<u>e</u> 0065	<u>f</u> 0066	<u>g</u> 0067	<u>h</u> 0068	<u>i</u> 0069	<u>j</u> 006A	<u>k</u> 006B	<u>l</u> 006C	<u>m</u> 006D	<u>n</u> 006E	<u>o</u> 006F
70	<u>p</u> 0070	<u>q</u> 0071	<u>r</u> 0072	<u>s</u> 0073	<u>t</u> 0074	<u>u</u> 0075	<u>v</u> 0076	<u>w</u> 0077	<u>x</u> 0078	<u>y</u> 0079	<u>z</u> 007A	<u>{</u> 007B	<u> </u> 007C	<u>}</u> 007D	<u>~</u> 007E	<u>DEL</u> 007F
80																
90																
A0	<u>NEBSP</u> 00A0	<u>†</u> 00A1	<u>‡</u> 00A2	<u>£</u> 00A3	<u>¤</u> 00A4	<u>¥</u> 00A5	<u>¦</u> 00A6	<u>§</u> 00A7	<u>¨</u> 00A8	<u>©</u> 00A9	<u>ª</u> 00AA	<u>«</u> 00AB	<u>¬</u> 00AC	<u>­</u> 00AD	<u>®</u> 00AE	<u>¯</u> 00AF
B0	<u>°</u> 00B0	<u>±</u> 00B1	<u>²</u> 00B2	<u>³</u> 00B3	<u>´</u> 00B4	<u>µ</u> 00B5	<u>¶</u> 00B6	<u>·</u> 00B7	<u>¸</u> 00B8	<u>¹</u> 00B9	<u>º</u> 00BA	<u>»</u> 00BB	<u>¼</u> 00BC	<u>½</u> 00BD	<u>¾</u> 00BE	<u>¿</u> 00BF
C0	<u>À</u> 00C0	<u>Á</u> 00C1	<u>Â</u> 00C2	<u>Ã</u> 00C3	<u>Ä</u> 00C4	<u>Å</u> 00C5	<u>Æ</u> 00C6	<u>Ç</u> 00C7	<u>È</u> 00C8	<u>É</u> 00C9	<u>Ê</u> 00CA	<u>Ë</u> 00CB	<u>Ì</u> 00CC	<u>Í</u> 00CD	<u>Î</u> 00CE	<u>Ï</u> 00CF
D0	<u>Ð</u> 00D0	<u>Ñ</u> 00D1	<u>Ò</u> 00D2	<u>Ó</u> 00D3	<u>Ô</u> 00D4	<u>Õ</u> 00D5	<u>Ö</u> 00D6	<u>×</u> 00D7	<u>Ø</u> 00D8	<u>Ù</u> 00D9	<u>Ú</u> 00DA	<u>Û</u> 00DB	<u>Ü</u> 00DC	<u>Ý</u> 00DD	<u>Þ</u> 00DE	<u>ß</u> 00DF
E0	<u>à</u> 00E0	<u>á</u> 00E1	<u>â</u> 00E2	<u>ã</u> 00E3	<u>ä</u> 00E4	<u>å</u> 00E5	<u>æ</u> 00E6	<u>ç</u> 00E7	<u>è</u> 00E8	<u>é</u> 00E9	<u>ê</u> 00EA	<u>ë</u> 00EB	<u>ì</u> 00EC	<u>í</u> 00ED	<u>î</u> 00EE	<u>ï</u> 00EF
F0	<u>ð</u> 00F0	<u>ñ</u> 00F1	<u>ò</u> 00F2	<u>ó</u> 00F3	<u>ô</u> 00F4	<u>õ</u> 00F5	<u>ö</u> 00F6	<u>÷</u> 00F7	<u>ø</u> 00F8	<u>ù</u> 00F9	<u>ú</u> 00FA	<u>û</u> 00FB	<u>ü</u> 00FC	<u>ý</u> 00FD	<u>þ</u> 00FE	<u>ÿ</u> 00FF

Figura 15: Página de Código ISO-8859-1.

## Páginas de Código do Windows

- **Visão Geral:**

- Múltiplas páginas de código para diferentes regiões e idiomas.
- Utilizadas no Microsoft Windows nas décadas de 1980 e 1990.

- **Exemplos:**

- CP1252 (Europa Ocidental), CP932 (Japão)
- CP1252 foi o sucessor do CP850 (utilizado no DOS).

- **Problemas:**

- Inconsistências entre diferentes sistemas.

## Transição de Codificação do Windows para Unicode

### ■ **UCS-2 (Conjunto de Caracteres Unicode - 2 bytes):**

- **Introdução:** Windows NT 3.1 (1993)

- **Detalhes:** Codificação de largura fixa de 16 bits para os primeiros 65.536 caracteres Unicode, usada internamente para APIs do Windows.

### ■ **UTF-16:**

- **Adoção:** Windows 2000 (2000)

- **Detalhes:** Uma extensão do UCS-2, acomodando todos os caracteres Unicode usando pares de substituição para caracteres além do Plano Multilíngue Básico (BMP).

- Exemplo de Par de Substituição: O emoji 😊 (Unicode U+1F60A) seria representado como: U+D83D (Substituto Alto) + U+DE0A (Substituto Baixo).

### ■ **UTF-8:**

- **Suporte Adicionado:** Windows 10 versão 1803 (Atualização de Abril de 2018)

- **Detalhes:** Codificação de largura variável, compatível com ASCII. Tornou-se mais amplamente suportada para desenvolvedores com a introdução da propriedade `ActiveCodePage` no Windows 10 versão 1903 (Atualização de Maio de 2019).

## Transição de Codificação do Windows para Unicode

### ■ **UCS-2 (Conjunto de Caracteres Unicode - 2 bytes):**

- **Introdução:** Windows NT 3.1 (1993)
- **Detalhes:** Codificação de largura fixa de 16 bits para os primeiros 65.536 caracteres Unicode, usada internamente para APIs do Windows.

### ■ **UTF-16:**

- **Adoção:** Windows 2000 (2000)
- **Detalhes:** Uma extensão do UCS-2, acomodando todos os caracteres Unicode usando pares de substituição para caracteres além do Plano Multilíngue Básico (BMP).
  - **Exemplo de Par de Substituição:** O emoji 😊 (Unicode U+1F60A) seria representado como: U+D83D (Substituto Alto) + U+DE0A (Substituto Baixo).

### ■ **UTF-8:**

- **Suporte Adicionado:** Windows 10 versão 1803 (Atualização de Abril de 2018)
- **Detalhes:** Codificação de largura variável, compatível com ASCII. Tornou-se mais amplamente suportada para desenvolvedores com a introdução da propriedade `ActiveCodePage` no Windows 10 versão 1903 (Atualização de Maio de 2019).

## Transição de Codificação do Windows para Unicode

### ■ **UCS-2 (Conjunto de Caracteres Unicode - 2 bytes):**

- **Introdução:** Windows NT 3.1 (1993)

- **Detalhes:** Codificação de largura fixa de 16 bits para os primeiros 65.536 caracteres Unicode, usada internamente para APIs do Windows.

### ■ **UTF-16:**

- **Adoção:** Windows 2000 (2000)

- **Detalhes:** Uma extensão do UCS-2, acomodando todos os caracteres Unicode usando pares de substituição para caracteres além do Plano Multilíngue Básico (BMP).

- Exemplo de Par de Substituição: O emoji 😊 (Unicode U+1F60A) seria representado como: U+D83D (Substituto Alto) + U+DE0A (Substituto Baixo).

### ■ **UTF-8:**

- **Suporte Adicionado:** Windows 10 versão 1803 (Atualização de Abril de 2018)

- **Detalhes:** Codificação de largura variável, compatível com ASCII. Tornou-se mais amplamente suportada para desenvolvedores com a introdução da propriedade `ActiveCodePage` no Windows 10 versão 1903 (Atualização de Maio de 2019).

# Unicode

- Conjunto de caracteres universal que abrange todos os scripts, suportando mais de 143.000 caracteres.
- Atribui um número único (chamado de “ponto de código”) a cada caractere, independentemente da plataforma, programa ou idioma.
- 1.112.064 pontos de código válidos dentro do espaço de códigos.
- A partir do Unicode 16.0, lançado em setembro de 2024, 299.056 (27%) desses pontos de código estão alocados, 155.063 (14%) têm caracteres atribuídos, 137.468 (12%) estão reservados para uso privado, 2.048 são usados para habilitar o mecanismo de substitutos, e 66 são designados como não-caracteres, deixando os restantes 815.056 (73%) não alocados.
- O Unicode possui diferentes formas de codificação: UTF-8, UTF-16 e UTF-32.

# Unicode

- Conjunto de caracteres universal que abrange todos os scripts, suportando mais de 143.000 caracteres.
- Atribui um número único (chamado de “ponto de código”) a cada caractere, independentemente da plataforma, programa ou idioma.
- 1.112.064 pontos de código válidos dentro do espaço de códigos.
- A partir do Unicode 16.0, lançado em setembro de 2024, 299.056 (27%) desses pontos de código estão alocados, 155.063 (14%) têm caracteres atribuídos, 137.468 (12%) estão reservados para uso privado, 2.048 são usados para habilitar o mecanismo de substitutos, e 66 são designados como não-caracteres, deixando os restantes 815.056 (73%) não alocados.
- O Unicode possui diferentes formas de codificação: UTF-8, UTF-16 e UTF-32.

# Unicode

- Conjunto de caracteres universal que abrange todos os scripts, suportando mais de 143.000 caracteres.
- Atribui um número único (chamado de “ponto de código”) a cada caractere, independentemente da plataforma, programa ou idioma.
- 1.112.064 pontos de código válidos dentro do espaço de códigos.
- A partir do Unicode 16.0, lançado em setembro de 2024, 299.056 (27%) desses pontos de código estão alocados, 155.063 (14%) têm caracteres atribuídos, 137.468 (12%) estão reservados para uso privado, 2.048 são usados para habilitar o mecanismo de substitutos, e 66 são designados como não-caracteres, deixando os restantes 815.056 (73%) não alocados.
- O Unicode possui diferentes formas de codificação: UTF-8, UTF-16 e UTF-32.



# Unicode

- Conjunto de caracteres universal que abrange todos os scripts, suportando mais de 143.000 caracteres.
- Atribui um número único (chamado de “ponto de código”) a cada caractere, independentemente da plataforma, programa ou idioma.
- 1.112.064 pontos de código válidos dentro do espaço de códigos.
- A partir do Unicode 16.0, lançado em setembro de 2024, 299.056 (27%) desses pontos de código estão alocados, 155.063 (14%) têm caracteres atribuídos, 137.468 (12%) estão reservados para uso privado, 2.048 são usados para habilitar o mecanismo de substitutos, e 66 são designados como não-caracteres, deixando os restantes 815.056 (73%) não alocados.
- O Unicode possui diferentes formas de codificação: UTF-8, UTF-16 e UTF-32.

# Unicode

- Conjunto de caracteres universal que abrange todos os scripts, suportando mais de 143.000 caracteres.
- Atribui um número único (chamado de “ponto de código”) a cada caractere, independentemente da plataforma, programa ou idioma.
- 1.112.064 pontos de código válidos dentro do espaço de códigos.
- A partir do Unicode 16.0, lançado em setembro de 2024, 299.056 (27%) desses pontos de código estão alocados, 155.063 (14%) têm caracteres atribuídos, 137.468 (12%) estão reservados para uso privado, 2.048 são usados para habilitar o mecanismo de substitutos, e 66 são designados como não-caracteres, deixando os restantes 815.056 (73%) não alocados.
- O Unicode possui diferentes formas de codificação: UTF-8, UTF-16 e UTF-32.

# UTF

## ■ UTF-8:

- Codificação de comprimento variável, compatível com ASCII, independente da ordem dos bytes.

## ■ UTF-16:

- Codificação de comprimento variável (2 ou 4 bytes por caractere).
- Caracteres latinos e os mais comumente usados CJK<sup>1</sup> são codificados em 2 bytes.

## ■ UTF-32:

- Codificação de comprimento fixo (4 bytes por caractere).

---

<sup>1</sup>Chinês, Japonês e Coreano.

# UTF

## ■ UTF-8:

- Codificação de comprimento variável, compatível com ASCII, independente da ordem dos bytes.

## ■ UTF-16:

- Codificação de comprimento variável (2 ou 4 bytes por caractere).
- Caracteres latinos e os mais comumente usados CJK<sup>1</sup> são codificados em 2 bytes.

## ■ UTF-32:

- Codificação de comprimento fixo (4 bytes por caractere).

---

<sup>1</sup>Chinês, Japonês e Coreano.

# UTF

- **UTF-8:**

- Codificação de comprimento variável, compatível com ASCII, independente da ordem dos bytes.

- **UTF-16:**

- Codificação de comprimento variável (2 ou 4 bytes por caractere).
- Caracteres latinos e os mais comumente usados CJK<sup>1</sup> são codificados em 2 bytes.

- **UTF-32:**

- Codificação de comprimento fixo (4 bytes por caractere).

---

<sup>1</sup>Chinês, Japonês e Coreano.

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

Figura 16: Estrutura do UTF-8.

## Domínio do UTF-8

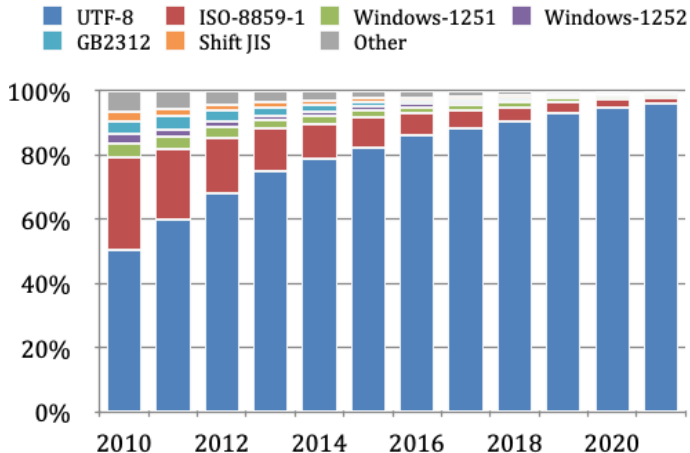


Figura 17: Conjunto de caracteres declarado para os 10 milhões de sites mais populares desde 2010.

# Endianness (estremicidade)

## ■ Big Endian vs. Little Endian:

- Ordem dos bytes na representação da memória.
- Impacta como os caracteres de múltiplos bytes são lidos.

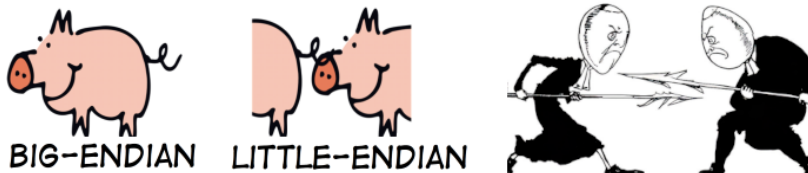
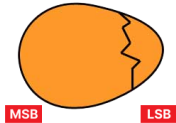


Figura 18: Endianness.



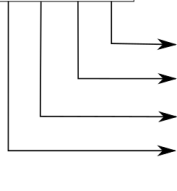
LITTLE ENDIAN - The Lilliputians break their eggs at the smaller end



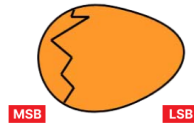
Little-endian

32-bit integer

0A0B0C0D



BIG ENDIAN - The Blefuscudians break their eggs at the big end.



Big-endian

32-bit integer

0A0B0C0D

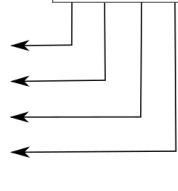


Figura 19: Big-Endian e Little-Endian.

## ■ Exemplo:

- UTF-16 e UTF-32 podem ser *big* ou *little endian*.
- Byte Order Mark (BOM) para indicar *endianness* em uso.
- O marcador BOM é o ponto de código U+FEFF (BOM, ZWNBSP<sup>2</sup>).
  - 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 no UTF-8: EF BB BF, funciona como uma assinatura para indicar que o arquivo está codificado em UTF-8 ao invés de especificar a ordem de bytes.

---

<sup>2</sup>zero width no-break space

## Formatos de Arquivos Texto

- **.txt:** Geralmente ASCII ou UTF-8.
- **.csv:** Pode usar várias codificações; importante para troca de dados.
- **.json:** Notação de objetos JavaScript, para intercâmbio de dados.
- **.yaml:** YAML não é linguagem de marcação, para serialização de dados.
- **.log:** Arquivos de log para registrar eventos, erros e atividades do sistema.
- **.ini:** Arquivos de inicialização para configurações.
- **.conf:** Arquivos de configuração, semelhantes aos .ini, usados por muitas aplicações.

# Markup Files

- **Markdown:**

- Linguagem de marcação leve para formatação de texto.
- O Markdown em si não possui um mecanismo embutido para declarar a codificação no cabeçalho do arquivo.

- **TeX:**

- Linguagem de composição tipográfica para tipografia de alta qualidade.
- Codificação: Frequentemente UTF-8, mas pode ser sensível a caracteres não-ASCII sem a configuração adequada do preâmbulo.
- `\usepackage[utf8]{inputenc}`

# Markup Files

- **Markdown:**

- Linguagem de marcação leve para formatação de texto.
- O Markdown em si não possui um mecanismo embutido para declarar a codificação no cabeçalho do arquivo.

- **TeX:**

- Linguagem de composição tipográfica para tipografia de alta qualidade.
- Codificação: Frequentemente UTF-8, mas pode ser sensível a caracteres não-ASCII sem a configuração adequada do preâmbulo.
- `\usepackage[utf8]{inputenc}`

- **XML (eXtensible Markup Language):**

- Usado para armazenamento e transmissão de dados estruturados.
- Codificação: Declarada na declaração XML, tipicamente UTF-8 ou UTF-16. A declaração de codificação é crucial para a análise correta.
- `<?xml version="1.0" encoding="UTF-8"?>`

- **HTML (HyperText Markup Language):**

- Linguagem de marcação padrão para documentos projetados para serem exibidos em um navegador da web.
- Codificação: O padrão é frequentemente UTF-8, mas pode ser especificado com o atributo `charset` na tag `<meta>`. Codificação incorreta pode levar a texto embaralhado.
- `<head><meta charset="UTF-8"></head>`
- Cabeçalho Content-Type HTTP: `Content-Type: text/html; charset=UTF-8`

- **XML (eXtensible Markup Language):**

- Usado para armazenamento e transmissão de dados estruturados.
- Codificação: Declarada na declaração XML, tipicamente UTF-8 ou UTF-16. A declaração de codificação é crucial para a análise correta.
- `<?xml version="1.0" encoding="UTF-8"?>`

- **HTML (HyperText Markup Language):**

- Linguagem de marcação padrão para documentos projetados para serem exibidos em um navegador da web.
- Codificação: O padrão é frequentemente UTF-8, mas pode ser especificado com o atributo `charset` na tag `<meta>`. Codificação incorreta pode levar a texto embaralhado.
- `<head><meta charset="UTF-8"></head>`
- Cabeçalho Content-Type HTTP: `Content-Type: text/html; charset=UTF-8`

## Ferramentas do Linux para Codificação de Texto

- **iconv:** Converte texto de uma codificação para outra.
  - Exemplo: `iconv -f ISO-8859-1 -t UTF-8 input.txt > output.txt`
- **file:** Identifica tipos de arquivos e codificações.
  - Exemplo: `file -i example.txt`
- **uconv (do ICU):** Conversão mais avançada com suporte a Unicode.
  - Exemplo: `uconv -f UTF-8 -t UTF-16 input.txt -o output.txt`
- **dos2unix / unix2dos:** Converte entre quebras de linha do Windows e do Unix.
  - Exemplo: `dos2unix file.txt` (converte CRLF para LF)
  - Exemplo: `unix2dos file.txt` (converte LF para CRLF)



## Ferramentas do Linux para Codificação de Texto

- **iconv:** Converte texto de uma codificação para outra.
  - Exemplo: `iconv -f ISO-8859-1 -t UTF-8 input.txt > output.txt`
- **file:** Identifica tipos de arquivos e codificações.
  - Exemplo: `file -i example.txt`
- **uconv (do ICU):** Conversão mais avançada com suporte a Unicode.
  - Exemplo: `uconv -f UTF-8 -t UTF-16 input.txt -o output.txt`
- **dos2unix / unix2dos:** Converte entre quebras de linha do Windows e do Unix.
  - Exemplo: `dos2unix file.txt` (converte CRLF para LF)
  - Exemplo: `unix2dos file.txt` (converte LF para CRLF)

## Ferramentas do Linux para Codificação de Texto

- **iconv:** Converte texto de uma codificação para outra.
  - Exemplo: `iconv -f ISO-8859-1 -t UTF-8 input.txt > output.txt`
- **file:** Identifica tipos de arquivos e codificações.
  - Exemplo: `file -i example.txt`
- **uconv (do ICU):** Conversão mais avançada com suporte a Unicode.
  - Exemplo: `uconv -f UTF-8 -t UTF-16 input.txt -o output.txt`
- **dos2unix / unix2dos:** Converte entre quebras de linha do Windows e do Unix.
  - Exemplo: `dos2unix file.txt` (converte CRLF para LF)
  - Exemplo: `unix2dos file.txt` (converte LF para CRLF)

## Ferramentas do Linux para Codificação de Texto

- **iconv:** Converte texto de uma codificação para outra.
  - Exemplo: `iconv -f ISO-8859-1 -t UTF-8 input.txt > output.txt`
- **file:** Identifica tipos de arquivos e codificações.
  - Exemplo: `file -i example.txt`
- **uconv (do ICU):** Conversão mais avançada com suporte a Unicode.
  - Exemplo: `uconv -f UTF-8 -t UTF-16 input.txt -o output.txt`
- **dos2unix / unix2dos:** Converte entre quebras de linha do Windows e do Unix.
  - Exemplo: `dos2unix file.txt` (converte CRLF para LF)
  - Exemplo: `unix2dos file.txt` (converte LF para CRLF)

## ■ **base64:**

- Exemplo: Usado para codificar anexos de e-mail.
- Uso: `echo "test" | base64` para codificar, `echo "dGVzdA==" | base64 -d` para decodificar.

## ■ **base58:**

- Exemplo: Útil para codificar endereços em criptomoedas (por exemplo, Bitcoin).
- Uso: `echo "test" | base58` para codificar, `echo "E8f4pE5" | base58 -d` para decodificar.

## ■ **base32:**

- Exemplo: Usado para codificar endereços no protocolo Segregated Witness (SegWit) do Bitcoin.
- Uso: `echo "test" | base32` para codificação, `echo "ORSXG5A=" | base32 -d` para decodificação.

## ■ **base64:**

- Exemplo: Usado para codificar anexos de e-mail.
- Uso: `echo "test" | base64` para codificar, `echo "dGVzdA==" | base64 -d` para decodificar.

## ■ **base58:**

- Exemplo: Útil para codificar endereços em criptomoedas (por exemplo, Bitcoin).
- Uso: `echo "test" | base58` para codificar, `echo "E8f4pE5" | base58 -d` para decodificar.

## ■ **base32:**

- Exemplo: Usado para codificar endereços no protocolo Segregated Witness (SegWit) do Bitcoin.
- Uso: `echo "test" | base32` para codificação, `echo "ORSXG5A=" | base32 -d` para decodificação.

## ■ **base64:**

- Exemplo: Usado para codificar anexos de e-mail.
- Uso: `echo "test" | base64` para codificar, `echo "dGVzdA==" | base64 -d` para decodificar.

## ■ **base58:**

- Exemplo: Útil para codificar endereços em criptomoedas (por exemplo, Bitcoin).
- Uso: `echo "test" | base58` para codificar, `echo "E8f4pE5" | base58 -d` para decodificar.

## ■ **base32:**

- Exemplo: Usado para codificar endereços no protocolo Segregated Witness (SegWit) do Bitcoin.
- Uso: `echo "test" | base32` para codificação, `echo "ORSXG5A=" | base32 -d` para decodificação.

## ■ **recode:**

- Função: Semelhante ao iconv, mas com capacidades adicionais.
- Uso: `recode latin1..utf-8 file.txt`

## ■ **xxd:**

- Função: Cria um despejo hexadecimal de um arquivo binário, útil para entender dados em nível de byte.
- Uso: `xxd -p file.bin` para hex simples, `xxd -r -p hex.txt` para reverter.

## ■ **Casos de Uso:**

- Migração de dados, limpeza e internacionalização.

## ■ **recode:**

- Função: Semelhante ao iconv, mas com capacidades adicionais.
- Uso: `recode latin1..utf-8 file.txt`

## ■ **xxd:**

- Função: Cria um despejo hexadecimal de um arquivo binário, útil para entender dados em nível de byte.
- Uso: `xxd -p file.bin` para hex simples, `xxd -r -p hex.txt` para reverter.

## ■ **Casos de Uso:**

- Migração de dados, limpeza e internacionalização.



- **recode:**

- Função: Semelhante ao iconv, mas com capacidades adicionais.
- Uso: `recode latin1..utf-8 file.txt`

- **xxd:**

- Função: Cria um despejo hexadecimal de um arquivo binário, útil para entender dados em nível de byte.
- Uso: `xxd -p file.bin` para hex simples, `xxd -r -p hex.txt` para reverter.

- **Casos de Uso:**

- Migração de dados, limpeza e internacionalização.

## Extensões de Arquivo

- **Apenas um Nome:** As extensões não definem o conteúdo do arquivo.
- **O Conteúdo Importa:** O tipo verdadeiro é determinado pelos dados internos.
- **Cuidado:** Extensões enganosas podem ser arriscadas.
- **Propósito:** Criadas para indicar o tipo de arquivo para facilitar o uso.

## Extensões de Arquivo

- **Apenas um Nome:** As extensões não definem o conteúdo do arquivo.
- **O Conteúdo Importa:** O tipo verdadeiro é determinado pelos dados internos.
- **Cuidado:** Extensões enganosas podem ser arriscadas.
- **Propósito:** Criadas para indicar o tipo de arquivo para facilitar o uso.

## Extensões de Arquivo

- **Apenas um Nome:** As extensões não definem o conteúdo do arquivo.
- **O Conteúdo Importa:** O tipo verdadeiro é determinado pelos dados internos.
- **Cuidado:** Extensões enganosas podem ser arriscadas.
- **Propósito:** Criadas para indicar o tipo de arquivo para facilitar o uso.

## Extensões de Arquivo

- **Apenas um Nome:** As extensões não definem o conteúdo do arquivo.
- **O Conteúdo Importa:** O tipo verdadeiro é determinado pelos dados internos.
- **Cuidado:** Extensões enganosas podem ser arriscadas.
- **Propósito:** Criadas para indicar o tipo de arquivo para facilitar o uso.

## Como o file Detecta a Codificação

### Números Mágicos / Assinaturas de Arquivo

- O file procura sequências de bytes específicas no início dos arquivos que identificam de forma única os formatos ou tipos de arquivo.
  - Para texto em UTF-8, procura o marcador BOM EF BB BF.
  - Para imagens JPEG, procura FF D8 FF.
  - Um arquivo PDF começa com %PDF.
  - Imagens PNG começam com os bytes 89 50 4E 47 0D 0A 1A 0A.
  - Arquivos WAV começam com 52 49 46 46.
  - Arquivos MP3 podem começar com 49 44 33.

## Como o file Detecta a Codificação

### Números Mágicos / Assinaturas de Arquivo

- O file procura sequências de bytes específicas no início dos arquivos que identificam de forma única os formatos ou tipos de arquivo.
  - Para texto em UTF-8, procura o marcador BOM EF BB BF.
  - Para imagens JPEG, procura FF D8 FF.
  - Um arquivo PDF começa com %PDF.
  - Imagens PNG começam com os bytes 89 50 4E 47 0D 0A 1A 0A.
  - Arquivos WAV começam com 52 49 46 46.
  - Arquivos MP3 podem começar com 49 44 33.

## Arquivos de Texto - Detecção de Codificação

- **Heurísticas:** Quando os números mágicos não são conclusivos, o `file` usa heurísticas.
  - **Análise de Caracteres:** Examina sequências de bytes em busca de padrões típicos de codificações específicas.
  - **Análise de Frequência:** Observa a frequência e a distribuição de caracteres para adivinhar o idioma e, assim, a codificação.
  - **Caracteres de Controle:** A presença ou ausência de certos caracteres de controle pode indicar a codificação.



## Arquivos de Texto - Detecção de Codificação

- **Heurísticas:** Quando os números mágicos não são conclusivos, o file usa heurísticas.
  - **Análise de Caracteres:** Examina sequências de bytes em busca de padrões típicos de codificações específicas.
  - **Análise de Frequência:** Observa a frequência e a distribuição de caracteres para adivinhar o idioma e, assim, a codificação.
  - **Caracteres de Controle:** A presença ou ausência de certos caracteres de controle pode indicar a codificação.

## Arquivos de Texto - Detecção de Codificação

- **Heurísticas:** Quando os números mágicos não são conclusivos, o file usa heurísticas.
  - **Análise de Caracteres:** Examina sequências de bytes em busca de padrões típicos de codificações específicas.
  - **Análise de Frequência:** Observa a frequência e a distribuição de caracteres para adivinhar o idioma e, assim, a codificação.
  - **Caracteres de Controle:** A presença ou ausência de certos caracteres de controle pode indicar a codificação.

## Arquivos de Texto - Detecção de Codificação

- **Heurísticas:** Quando os números mágicos não são conclusivos, o `file` usa heurísticas.
  - **Análise de Caracteres:** Examina sequências de bytes em busca de padrões típicos de codificações específicas.
  - **Análise de Frequência:** Observa a frequência e a distribuição de caracteres para adivinhar o idioma e, assim, a codificação.
  - **Caracteres de Controle:** A presença ou ausência de certos caracteres de controle pode indicar a codificação.

## Banco de Dados MIME (Multipurpose Internet Mail Extension)

- Mapeia o conteúdo do arquivo para tipos MIME e codificações.
- Localização: `/usr/share/file/magic.mgc` ou similar (banco de dados compilado).
  - O banco de dados `magic.mgc` é gerado a partir de um conjunto de arquivos de texto “mágicos” (por exemplo, `/etc/magic`) que definem as regras para reconhecer vários formatos de arquivo.
  - As regras consistem em:
    - Deslocamentos de byte
    - Padrões de byte
    - Expressões regulares
    - Descrições legíveis por humanos
  - Exemplo:

```
0  string  \x89PNG\r\n\x1a\n  PNG image data
0  string  %PDF-                PDF document
```

[Lista de assinaturas de arquivos \(Wikipedia\)](#)

## Banco de Dados MIME (Multipurpose Internet Mail Extension)

- Mapeia o conteúdo do arquivo para tipos MIME e codificações.
- Localização: `/usr/share/file/magic.mgc` ou similar (banco de dados compilado).
  - O banco de dados `magic.mgc` é gerado a partir de um conjunto de arquivos de texto “mágicos” (por exemplo, `/etc/magic`) que definem as regras para reconhecer vários formatos de arquivo.
  - As regras consistem em:
    - Deslocamentos de byte
    - Padrões de byte
    - Expressões regulares
    - Descrições legíveis por humanos

- Exemplo:

```
0  string  \x89PNG\r\n\x1a\n  PNG image data
0  string  %PDF-                PDF document
```

## Lista de assinaturas de arquivos (Wikipedia)

## Banco de Dados MIME (Multipurpose Internet Mail Extension)

- Mapeia o conteúdo do arquivo para tipos MIME e codificações.
- Localização: `/usr/share/file/magic.mgc` ou similar (banco de dados compilado).
  - O banco de dados `magic.mgc` é gerado a partir de um conjunto de arquivos de texto “mágicos” (por exemplo, `/etc/magic`) que definem as regras para reconhecer vários formatos de arquivo.
  - As regras consistem em:
    - Deslocamentos de byte
    - Padrões de byte
    - Expressões regulares
    - Descrições legíveis por humanos
  - Exemplo:

0	string	<code>\x89PNG\r\n\x1a\n</code>	PNG image data
0	string	<code>%PDF-</code>	PDF document

Lista de assinaturas de arquivos (Wikipedia)

## Reconhecimento/Falsas Confusões

- **Ambiguidade:** Alguns arquivos podem ser interpretados como múltiplas codificações, especialmente se contiverem apenas caracteres ASCII.
  - Exemplo: Um arquivo com apenas ASCII pode ser relatado como `us-ascii`, mas pode ser UTF-8 ou ISO-8859-1.
- **Informação Incompleta:** Arquivos curtos ou arquivos com um conjunto de caracteres limitado podem não fornecer dados suficientes para uma detecção precisa.
- **Sobreposição de Codificação:** Codificações que compartilham um subconjunto de caracteres (como ASCII em UTF-8) podem levar a confusões.
- **Binário em Texto:** Arquivos com dados binários incorporados podem confundir a ferramenta, fazendo-a pensar que é um arquivo binário em vez de texto com codificação.
- **Falsos Positivos:** Às vezes, o `file` pode errar na suposição devido a padrões que imitam outra codificação ou devido a um banco de dados mágico atualizado, mas não abrangente.

## Reconhecimento/Falsas Confusões

- **Ambiguidade:** Alguns arquivos podem ser interpretados como múltiplas codificações, especialmente se contiverem apenas caracteres ASCII.
  - Exemplo: Um arquivo com apenas ASCII pode ser relatado como `us-ascii`, mas pode ser UTF-8 ou ISO-8859-1.
- **Informação Incompleta:** Arquivos curtos ou arquivos com um conjunto de caracteres limitado podem não fornecer dados suficientes para uma detecção precisa.
- **Sobreposição de Codificação:** Codificações que compartilham um subconjunto de caracteres (como ASCII em UTF-8) podem levar a confusões.
- **Binário em Texto:** Arquivos com dados binários incorporados podem confundir a ferramenta, fazendo-a pensar que é um arquivo binário em vez de texto com codificação.
- **Falsos Positivos:** Às vezes, o `file` pode errar na suposição devido a padrões que imitam outra codificação ou devido a um banco de dados mágico atualizado, mas não abrangente.



## Reconhecimento/Falsas Confusões

- **Ambiguidade:** Alguns arquivos podem ser interpretados como múltiplas codificações, especialmente se contiverem apenas caracteres ASCII.
  - Exemplo: Um arquivo com apenas ASCII pode ser relatado como `us-ascii`, mas pode ser UTF-8 ou ISO-8859-1.
- **Informação Incompleta:** Arquivos curtos ou arquivos com um conjunto de caracteres limitado podem não fornecer dados suficientes para uma detecção precisa.
- **Sobreposição de Codificação:** Codificações que compartilham um subconjunto de caracteres (como ASCII em UTF-8) podem levar a confusões.
- **Binário em Texto:** Arquivos com dados binários incorporados podem confundir a ferramenta, fazendo-a pensar que é um arquivo binário em vez de texto com codificação.
- **Falsos Positivos:** Às vezes, o `file` pode errar na suposição devido a padrões que imitam outra codificação ou devido a um banco de dados mágico atualizado, mas não abrangente.

## Reconhecimento/Falsas Confusões

- **Ambiguidade:** Alguns arquivos podem ser interpretados como múltiplas codificações, especialmente se contiverem apenas caracteres ASCII.
  - Exemplo: Um arquivo com apenas ASCII pode ser relatado como `us-ascii`, mas pode ser UTF-8 ou ISO-8859-1.
- **Informação Incompleta:** Arquivos curtos ou arquivos com um conjunto de caracteres limitado podem não fornecer dados suficientes para uma detecção precisa.
- **Sobreposição de Codificação:** Codificações que compartilham um subconjunto de caracteres (como ASCII em UTF-8) podem levar a confusões.
- **Binário em Texto:** Arquivos com dados binários incorporados podem confundir a ferramenta, fazendo-a pensar que é um arquivo binário em vez de texto com codificação.
- **Falsos Positivos:** Às vezes, o `file` pode errar na suposição devido a padrões que imitam outra codificação ou devido a um banco de dados mágico atualizado, mas não abrangente.

## Reconhecimento/Falsas Confusões

- **Ambiguidade:** Alguns arquivos podem ser interpretados como múltiplas codificações, especialmente se contiverem apenas caracteres ASCII.
  - Exemplo: Um arquivo com apenas ASCII pode ser relatado como `us-ascii`, mas pode ser UTF-8 ou ISO-8859-1.
- **Informação Incompleta:** Arquivos curtos ou arquivos com um conjunto de caracteres limitado podem não fornecer dados suficientes para uma detecção precisa.
- **Sobreposição de Codificação:** Codificações que compartilham um subconjunto de caracteres (como ASCII em UTF-8) podem levar a confusões.
- **Binário em Texto:** Arquivos com dados binários incorporados podem confundir a ferramenta, fazendo-a pensar que é um arquivo binário em vez de texto com codificação.
- **Falsos Positivos:** Às vezes, o `file` pode errar na suposição devido a padrões que imitam outra codificação ou devido a um banco de dados mágico atualizado, mas não abrangente.

## Conclusão

### ■ Pontos Chave:

- **Jornada Através da Codificação:** Desde códigos históricos como Morse e Baudot até padrões modernos como UTF-8 e Unicode.
- **Evolução:** A codificação de texto passou de sistemas simples para sistemas complexos.
- **Solução Universal:** O Unicode fornece uma representação global de texto.
- **Conceitos, Desafios, Soluções, Aplicações e Consciência:** Compreender esses aspectos é crucial.

### ■ Futuro:

- Evolução contínua dos padrões de codificação para acomodar novos scripts e símbolos.

