

**NAME**

UTF-8 - an ASCII compatible multibyte Unicode encoding

**DESCRIPTION**

The Unicode 3.0 character set occupies a 16-bit code space. The most obvious Unicode encoding (known as UCS-2) consists of a sequence of 16-bit words. Such strings can contain—as part of many 16-bit characters—bytes such as `'\0'` or `'/'`, which have a special meaning in filenames and other C library function arguments. In addition, the majority of UNIX tools expect ASCII files and can't read 16-bit words as characters without major modifications. For these reasons, UCS-2 is not a suitable external encoding of Unicode in filenames, text files, environment variables, and so on. The ISO 10646 Universal Character Set (UCS), a superset of Unicode, occupies an even larger code space—31 bits—and the obvious UCS-4 encoding for it (a sequence of 32-bit words) has the same problems.

The UTF-8 encoding of Unicode and UCS does not have these problems and is the common way in which Unicode is used on UNIX-style operating systems.

**Properties**

The UTF-8 encoding has the following nice properties:

- \* UCS characters `0x00000000` to `0x0000007f` (the classic US-ASCII characters) are encoded simply as bytes `0x00` to `0x7f` (ASCII compatibility). This means that files and strings which contain only 7-bit ASCII characters have the same encoding under both ASCII and UTF-8.
- \* All UCS characters greater than `0x7f` are encoded as a multibyte sequence consisting only of bytes in the range `0x80` to `0xfd`, so no ASCII byte can appear as part of another character and there are no problems with, for example, `'\0'` or `'/'`.
- \* The lexicographic sorting order of UCS-4 strings is preserved.
- \* All possible  $2^{31}$  UCS codes can be encoded using UTF-8.
- \* The bytes `0xc0`, `0xc1`, `0xfe`, and `0xff` are never used in the UTF-8 encoding.
- \* The first byte of a multibyte sequence which represents a single non-ASCII UCS character is always in the range `0xc2` to `0xfd` and indicates how long this multibyte sequence is. All further bytes in a multibyte sequence are in the range `0x80` to `0xbf`. This allows easy resynchronization and makes the encoding stateless and robust against missing bytes.
- \* UTF-8 encoded UCS characters may be up to six bytes long, however the Unicode standard specifies no characters above `0x10ffff`, so Unicode characters can be only up to four bytes long in UTF-8.

**Encoding**

The following byte sequences are used to represent a character. The sequence to be used depends on the UCS code number of the character:

`0x00000000` - `0x0000007F`:  
`0xxxxxx`

`0x00000080` - `0x000007FF`:  
`110xxxx 10xxxxx`

`0x00000800` - `0x0000FFFF`:  
`1110xxx 10xxxxx 10xxxxx`

`0x00010000` - `0x001FFFFF`:  
`11110xxx 10xxxxx 10xxxxx 10xxxxx`

`0x00200000` - `0x03FFFFFF`:  
`111110xx 10xxxxx 10xxxxx 10xxxxx 10xxxxx`

`0x04000000` - `0x7FFFFFFF`:  
`1111110x 10xxxxx 10xxxxx 10xxxxx 10xxxxx 10xxxxx`

The `xxx` bit positions are filled with the bits of the character code number in binary representation, most significant bit first (big-endian). Only the shortest possible multibyte sequence which can represent the

code number of the character can be used.

The UCS code values 0xd800–0xdfff (UTF-16 surrogates) as well as 0xfffe and 0xffff (UCS noncharacters) should not appear in conforming UTF-8 streams. According to RFC 3629 no point above U+10FFFF should be used, which limits characters to four bytes.

### Example

The Unicode character 0xa9 = 1010 1001 (the copyright sign) is encoded in UTF-8 as

```
11000010 10101001 = 0xc2 0xa9
```

and character 0x2260 = 0010 0010 0110 0000 (the "not equal" symbol) is encoded as:

```
11100010 10001001 10100000 = 0xe2 0x89 0xa0
```

### Application notes

Users have to select a UTF-8 locale, for example with

```
export LANG=en_GB.UTF-8
```

in order to activate the UTF-8 support in applications.

Application software that has to be aware of the used character encoding should always set the locale with for example

```
setlocale(LC_CTYPE, "")
```

and programmers can then test the expression

```
strcmp(nl_langinfo(CODESET), "UTF-8") == 0
```

to determine whether a UTF-8 locale has been selected and whether therefore all plaintext standard input and output, terminal communication, plaintext file content, filenames and environment variables are encoded in UTF-8.

Programmers accustomed to single-byte encodings such as US-ASCII or ISO 8859 have to be aware that two assumptions made so far are no longer valid in UTF-8 locales. Firstly, a single byte does not necessarily correspond any more to a single character. Secondly, since modern terminal emulators in UTF-8 mode also support Chinese, Japanese, and Korean double-width characters as well as nonspacing combining characters, outputting a single character does not necessarily advance the cursor by one position as it did in ASCII. Library functions such as [mbsrtowcs\(3\)](#) and [wcswidth\(3\)](#) should be used today to count characters and cursor positions.

The official ESC sequence to switch from an ISO 2022 encoding scheme (as used for instance by VT100 terminals) to UTF-8 is ESC % G ("\x1b%G"). The corresponding return sequence from UTF-8 to ISO 2022 is ESC % @ ("\x1b%@"). Other ISO 2022 sequences (such as for switching the G0 and G1 sets) are not applicable in UTF-8 mode.

### Security

The Unicode and UCS standards require that producers of UTF-8 shall use the shortest form possible, for example, producing a two-byte sequence with first byte 0xc0 is nonconforming. Unicode 3.1 has added the requirement that conforming programs must not accept non-shortest forms in their input. This is for security reasons: if user input is checked for possible security violations, a program might check only for the ASCII version of "/./" or ";" or NUL and overlook that there are many non-ASCII ways to represent these things in a non-shortest UTF-8 encoding.

### Standards

ISO/IEC 10646-1:2000, Unicode 3.1, RFC 3629, Plan 9.

### SEE ALSO

[locale\(1\)](#), [nl\\_langinfo\(3\)](#), [setlocale\(3\)](#), [charsets\(7\)](#), [unicode\(7\)](#)

### COLOPHON

This page is part of release 4.10 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.