Codages de caractères
ASCII, latin-1, UTF-8, etc.

S. Aicardi

Journées Mathrice, Nantes, 14-16 Mars 2006

Problèmes de codages

Le réseau internet permet de faire de choses incroyables : un mail peut être quasi-instantanément envoyé à l'autre bout du monde en passant par des serveurs de marques différentes, de systèmes d'exploitation différents, de Mail Transfer Agent différents et être pourtant lu sans problème par le destinataire.

Sauf que parfois un

Mathrice, c'est génial

peut se transformer en

Mathrice, c'est gÈnial

ou en

Mathrice, c'est gÃ©nial

Au début était l'ASCII

La norme ASCII est publiée pour la première fois en 1963 et à peu près stabilisée dans sa version actuelle en 1967.

Le codage ASCII est dérivé des codages utilisé en télégraphie. À cette époque, la bande passante est faible, la qualité de transmission est aléatoire. Sur les 8 bits destinés à coder une information, 7 sont utilisés pour les données et un pour la parité.

La table des 128 caractères est divisée en groupes de 32.

- des caractères de contrôles, pour la plupart obsolètes.
- les chiffres et la plupart des signes de ponctuation.
- les majuscules.
- les minuscules.

Table des caractères ASCII

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
BS	HT	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
CAN	EM	SUB	ESC	FS	GS	RS	US
	!	"	#	$	%	&	'
(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7
8	9	:	;	<	=	>	?
@	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	{	|	}	~	DEL

Les caractères de contrôle

Ils servaient à piloter une connexion ou une imprimante. Certains sont restés :

- NUL : fin de chaine de caractères dans de nombreux langages de programmation. À l'origine, caractère qui ne fait rien.
- EOT ou ^D : sert de fin de transmission (cf la commande unix `cat`).
- BEL ou ^G : fait beeep !
- BS ou ^H : backspace. Était utilisé pour écrire les accents par dessus les lettres.
- DEL : delete.
- FF ou ^L : initialement passe à la page d'après. Souvent implémenté en nettoyage de l'écran.
- ESC : escape. Début de séquences de formattage.
- LF et CR : voir plus loin

Les limitations de l'ASCII

- de nombreux caractères de contrôle inutilisés,
- pas de définition précise des caractères de contrôle (exemple : LF et CR),
- limité à l'alphabet latin sans accents,
- en 7 bits, donc facilement extensible !

Premier étage de la tour de Babel

En passant des systèmes de transmission aux systèmes d'exploitation, la plupart des caractères de contrôle de transfert perdent leur sens. Par exemple, pour finir une ligne, un fichier va à la ligne. Une imprimante ramène le charriot en début de ligne (CR) et passe à la ligne suivante (LF).

La suite de caractères CRLF est restée en standard de fin de ligne dans certains protocoles réseaux (SMTP, HTTP, FTP...). Par ailleurs, des systèmes comme CP/M, DOS et Windows ont conservé cet usage pour les fichiers.

Depuis Multics jusqu'à toutes les variantes Unix, cette suite a été simplifiée en LF sous le nom *newline*.

Apple a également simplifié la convention de fin de ligne, mais en gardant CR, en tous cas jusqu'à OS 9. OS X choisit nativement le format Unix, mais reconnait l'ancien format Mac.

Quelques traducteurs

- `dos2unix`, `unix2dos`
- `mac2unix`, `unix2mac`
- d'innombrables solutions à coup de `sed`, `tr`, `perl`, etc.
- la plupart des éditeurs permettent de le faire de manière transparente.

Jusqu'ici, on arrive à peu près à communiquer.

Extensions à 8 bits

Le passage à l'écran supprime la possibilité de composer des caractères et des accents. Les langues d'Europe Occidentale deviennent impossible à rendre sur un écran avec seulement la norme ASCII 7 bits.

Par ailleurs, toutes les alphabets autres que l'alphabet latin ne sont pas gérés par l'ASCII.

Le progrès des technologies permet d'utiliser le huitième bit des octets.

$\Longrightarrow$ Chaque langue se voit dotée d'une extension de la norme ASCII pour coder son alphabet.

En Asie

Dès 1969, *JIS X 0201* permet de coder une partie du japonais sur un octet. Le codage n'est pas 100% compatible avec ASCII : le \ est remplacé par le ¥, et le ~ est remplacé par le ‾.
Afin de rendre tous les caractères japonais, on passe à un codage sur deux octets avec plusieurs normes distinctes : *Shift_JIS*, *ISO-2022-JP*, *EUC-JP*.

Standardisé en 1980, *GB 2312* code le chinois simplifié sur deux octets de manière plus compacte qu'*UTF-8*. Taïwan et HongKong qui utilisent un chinois non simplifié adoptent le *Big5*.

Globalement, pour chaque alphabet asiatique, on a un minimum de deux normes distinctes : *ISO-2022* et *EUC* (Extended Unix Code).

En Europe Occidentale : ISO-8859 et autres

Pour coder sur 8 bits les accents des langues européennes, plusieurs encodages sont apparus :

- *IBM code page 437, puis 850* sur IBM PC en 1981. Partiellement compatible avec ASCII.
- *ISO-8859-1* ou *Latin-1*. Compatible avec ASCII et réserve de la place pour de nouveaux caractères de contrôle.
- *Windows-1252*. Extension de ISO-8859-1 qui utilise la zone réservée aux caractères de contrôle.
- *ISO-8859-15* ou *Latin-9*. Essentiellement identique à ISO-8859-1, avec quelques remplacements pour compléter le support du français, du finnois et de l'estonien.
- *Mac Roman*. Strictement incompatible avec ISO-8859-1 ou les codes pages IBM.

Table des caractères Windows-1252 et de ISO-8859-15

Voici la liste des caractères à partir du 128e. En rouge, les caractères de ISO-8859-15 qui diffèrent de ISO-8859-1. En bleu, les caractères de contrôle de ISO-8859-1.

€ PAD	HOP	‚ BPH	f NBH	„ IND	… NEL	† SSA	‡ ESA
ˆ HTS	‰ HTJ	Š VTS	‹ PLD	Œ PLU	RI	Ž SS2	SS3
DCS	‘ PU1	’ PU2	“ STS	” CCH	• MW	– SPA	— EPA
˜ SOS	™ SGCI	š SCI	› CSI	œ ST	OSC	ž PM	Ÿ APC
NBSP	¡	¢	£	¤ €	¥	¦ Š	§
¨ š	©	ª	«	¬	SHY	®	¯
°	±	2	3	´ Ž	µ	¶	·
¸ ž	1	º	»	¼ Œ	½ œ	¾ Ÿ	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç
È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×
Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç
è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷
ø	ù	ú	û	ü	ý	þ	ÿ

En Europe Orientale : ISO-8859 et autres

Pour l'alphabet cyrillique, trois codages mutuellement incompatibles sont utilisés : *ISO-8559-5*, *Windows-1251* et *KOI-R* (*KOI-U* pour l'Ukrainien).

Pour l'alphabet grec (monotonal), la norme principale est *ISO-8859-7*, mais il est couvert également par *Mac Roman*.

Deuxième étage de la Tour de Babel

Sans méta-donnée supplémentaire, il est impossible de savoir informatiquement si un texte contenant du 8 bits est encodé en Latin-1, en Latin-9, en Mac Roman, en Cyrillique ISO, Windows ou KOI ou dans une langue asiatique.

Comment convertir ? C'est très difficile car les incompatibilités sont souvent mutuelles (pas de € dans ISO-8859-1, pas de ½ dans ISO-8859-15).

Quelques traducteurs

La commande `file` permet parfois d'identifier l'encodage des fichiers.

Les commandes `iconv` et `recode` permettent de convertir la plupart des locales (quand on connait l'encodage d'origine).

Unicode

Entre les langues qui ne pouvaient pas s'écrire dans une norme ISO ou propriétaire et celles qui se perdaient dans les conflits de formats, sans compter l'impossibilité de fichiers multilingues, il y avait un besoin d'un standard universel comme ASCII pour permettre les échanges numériques.

Ce besoin est à peu près rempli par *Unicode* et *ISO-10646*.

La première édition d'Unicode date de 1991. Les standards Unicode et ISO-10646 sont identiques depuis 1993. La version actuelle est 4.1.

Unicode Consortium

Unicode est publié par l'*Unicode Consortium*, une organisation à but non lucratif.

Quelques membres : Adobe, Apple, Google, HP, IBM, Microsoft, Oracle, Sun, Verisign, Yahoo!

On peut donc espérer obtenir un standard largement diffusé !

Unicode

Unicode fournit un standard unique pour représenter une immense majorité des alphabets actuels et anciens.

Les caractères sont codés sur 21 bits. Notation : `U+xxxxxx`.

Exemples :

- `U+0041` : A LATIN CAPITAL LETTER A.
- `U+00C7` : Ç LATIN CAPITAL LETTER C WITH CEDILLA.
- `U+03D0` : β GREEK BETA SYMBOL
- `U+2200` : $\forall$ FOR ALL
- `U+FB01` : ﬁ LATIN SMALL LIGATURE FI
- `U+1D15F` : 𝅗𝅥 MUSICAL SYMBOL QUARTER NOTE

Unicode

Le codage unicode est divisé en 17 plans de 2^{16} caractères. Le premier (U+0000 à U+FFFF) est appelé “Basic Multilingual Plane”. Il contient la plupart des langues modernes et un certain nombre de symboles. Il est amplement suffisant pour les communications internationales standard.

Exemples :

- La plage U+0000 à U+00FF est identique à ISO-8859-1
- la plage U+0370 à U+03FF consacrée à l'alphabet grec reprend ISO-8859-7.
- la plage U+0400 à U+045F consacrée à l'alphabet cyrillique russe reprend pour l'essentiel ISO-8859-5.

Caractères vs glyphes

De nombreux autres codages existants ont été repris tels quels. On obtient donc un certain nombre de caractères ayant même représentation graphique (*glyphe*).

Par exemple, le caractère `U+0041` s'appelle LATIN CAPITAL LETTER A et se représente A, de même que le caractère `U+0391` qui s'appelle GREEK CAPITAL LETTER ALPHA, ou `U+0410` qui s'appelle CYRILLIC CAPITAL LETTER A.

Unicode introduit une différence entre le caractère et sa représentation graphique

Caractères vs glyphes

Cas particulier, certaines représentations graphiques peuvent s'écrire en plus d'un caractère unicode.

Exemple : é peut se représenter par le caractère `U+00E9`, ou se représenter par `U+0065` (e) suivi de `U+0301` (́).

De même, Å peut se représenter par `U+00C5` (A with ring above), par `U+0041` (A) suivi de `U+030A` (̊), ou par `U+212B` (Angstrom sign).

Comment doit-on stocker dans un fichier ces caractères ?

Normalisations

Unicode définit des opérations de composition et de décomposition sur certains caractères.

Il y a essentiellement deux formes normalisées :

- la forme NFD obtenue après décomposition maximale,
- la forme NFC obtenue après décomposition maximale puis recomposition

Pour beaucoup plus de détails, voir l'annexe 15 d'Unicode.

Encodages d'Unicode

Unicode est pour l'instant codé en 21 bits, mais les ordinateurs raisonnent plutôt en multiples de 8 bits. Il faut donc transformer le code Unicode en quelque chose de plus utilisable.

Il existe 5 standards supportés par l'Unicode Consortium ou par l'ISO.

- UTF-8 (tout encodé sur des unités de 8 bits)
- UTF-16 (tout encodé sur des unités de 16 bits)
- UCS-2 (la BMP encodée sur 16 bits)
- UTF-32=UCS-4 (tout encodé sur 32 bits)

d'autres encodages sont possibles : UTF-7, SCSU,...

UTF-8

Principe :

de U+0000 à U+007F	0vvvvvvv
de U+0080 à U+07FF	110vvvvv 10vvvvvv
de U+0800 à U+FFFF	1110vvvv 10vvvvvv 10vvvvvv
de U+10000 à U+1FFFFF	11110vvv 10vvvvvv 10vvvvvv 10vvvvvv

Exemples :

A	U+0041	0100001
Ç	U+00C7	11000011 10000111
β	U+03D0	11001111 10010000
$\forall$	U+2200	11100010 10001000 10000000

UTF-16

Principe :

de U+0000 à U+FFFF	vvvvvvvv vvvvvvvv (big endian)
de U+0000 à U+FFFF	vvvvvvvv vvvvvvvv (little endian)
de U+010000 à U+10FFFF	110110uu uuvvvvvv 110111vv vvvvvvvv (big)
de U+010000 à U+10FFFF	uuvvvvvv 110110uu vvvvvvvv 110111vv (little)

où uuuu est la représentation binaire du nombre en rouge $\underline{-1}$

Exemples (en big endian) :

Ç	U+00C7	00000000 11000111
𝅗𝅥	U+01D15F	11011000 00110100 11011101 01011111

Remarque : Pour éviter des conflits dans la notation de UTF-16, les caractères de U+D800 à U+DFFF ne sont pas attribués.

Gros vs Petits-boutistes

UTF-16 semble introduire une incompatibilité entre les machines *big endian* et les machines *little endian*.

En fait, non : il y trois UTF-16 : UTF-16LE (little endian), UTF-16BE (big endian) et UTF-16.

Le troisième impose de commencer un texte par un “Byte Order Mark”, le caractère unicode non affichable U+FEFF (Zero-Width No-Break Space). Si le texte reçu commence par `FE FF`, on est en UTF-16 big endian, s'il commence par `FF FE`, on est en UTF-16 little endian.

Très gros avantage de cette technique : on peut reconnaître informatiquement un texte en UTF-16.

Les codages par défaut selon les OS

Linux : UTF-8 est utilisé dans les locales par défaut dans la plupart des distributions récentes.

Windows : UTF-8 est supporté depuis Windows 2000.

Mac : Depuis OS X, UTF-8 NFD est utilisé pour les noms de fichiers. Les fichiers de configuration (.plist) sont en UTF-16. Le reste est au choix du client (Mac OS Roman par défaut dans TextEdit si on tape du Français, Iso-8859-1 dans le terminal, etc.).

Le retour de la vengeance de la tour de Babel

Tant que Unicode et ses formats de transformation ne sont pas définitivement utilisés de part le monde, l'apport d'Unicode est d'avoir un risque d'incompatibilité en plus.

De nombreux programmes Windows (par ex: NotePad) utilisent la technique du Byte Order Mark pour indiquer l'UTF-8 (`EF BB BF`). Les Unix-like s'y refusent (à cause du `#!/bin/sh`).

Plus génant, les bureaux Unix et Windows utilisent de préférence UTF-8 NFC pour coder les noms de fichiers, Mac OS X impose UTF-8 NFD.

Faut-il migrer vers Unicode ?

Oui[1]

Tout de suite, si c'est possible. Typiquement, on n'a pas une grosse quantité de fichiers installés ou si bon nombre d'utilisateurs fonctionnent déjà en Unicode (grâce à leur portable).

Prévoir une migration à relativement court terme, même si on a des Tera-Octets de documents en Latin-1. Ca ne peut qu'empirer et le ratio ISO-8859-1 vs Unicode devient tous les jours plus défavorable.

[1] Disclaimer : C'est un avis pas encore mis en pratique !

Vers quel UTF ?

Si on manipule majoritairement des Langues Européennes, de l'arabe et de l'hébreu, UTF-8 est le meilleur compromis place disque/nombre de caractères codés.

Gros avantage : UTF-8 est le seul format de transformation directement compatible avec ASCII.

Si le chinois ou le japonais sont utilisés, UTF-16 est plus économique.

Comment migrer vers UTF-8 ?

Une bonne partie du boulot sera à la charge des utilisateurs, donc bien préparer le terrain.

On peut migrer automatiquement :

- les noms de fichiers avec convmv
- les fichiers textes avec iconv ou recode (mais il faut s'assurer de ce qu'on fait)
- Pour les fichiers TeX, penser à remplacer `\usepackage[latin1]{inputenc}` par `\usepackage[utf8]{inputenc}` ou par `\usepackage[utf8x]{inputenc}`.
  Mais beaucoup d'utilisateurs font du TeX en 7 bits !
- Pour les serveurs Web, mettre un AddDefaultCharset UTF-8 dans la config d'Apache.

Comment choisir son encodage ?

`export LANG=fr_FR.utf-8` dans le `.bash_profile` est une solution.

On peut choisir ponctuellement l'encodage d'un terminal :

- pour xterm, il faut le lancer par
  `LANG=fr_FR.UTF-8 xterm -lc &`
- pour les terminaux de Gnome ou de KDE, il y a l'option dans les menus.

Pour la plupart des applications, il faudra regarder au cas par cas.

References

Wikipedia, en particulier en version anglaise, fait un très bon résumé

La page web de l'Unicode Consortium contient les standards et de nombreux documents techniques.