
4 Lokalisierung und Internationalisierung

In diesem Kapitel lernen Sie

- ▶ ein Linux-System für einen Benutzer in einer anderen Sprache einzurichten.

Dadurch, dass das Internet die Erde zu einem „globalen Dorf“ vernetzt hat und viele Unternehmen Zweigstellen in mehreren Ländern und Sprachräumen betreiben, hat die Internationalisierung und Lokalisierung von Linux stark an Bedeutung gewonnen, insbesondere da Linux nun im Begriff ist, zunehmend als Desktop-System benutzt zu werden (LPI 1: 107.3).



Internationalisierung und Lokalisierung werden wegen der Länge der beiden Wörter oft mit *i18n* und *l10n* abgekürzt. (Die Zahlen 18 und 10 entsprechen hier der Anzahl der weggelassenen Buchstaben in den englischen Wörtern ‘internationalization’ und ‘localization’).

4.1 Lokalisierung der Systemzeit

/usr/share/zoneinfo/: Dieses Verzeichnis enthält die Zeitumrechnungsrou-tinen (Winter-/Sommerzeit, Unterschied zur UTC) zu allen Zeitzonen der Welt (LPI 1: 107.3). Für jede Zeitzone existiert eine zugehörige Datei, wie z.B. `/usr/share/zoneinfo/Europe/Berlin` für Deutschland.



/etc/timezone: Diese Datei enthält den Namen der lokalen Zeitzone und hat wenig Auswirkung auf das System (LPI 1: 107.3); sie sollte trotzdem nicht von Hand editiert werden. Sie wird vor allem in Debian Linux genutzt. Red-Hat/Fedora dagegen verwendet `/etc/sysconfig/clock`.



/etc/localtime: Binäre Datei mit den Umrechnungsrou-tinen zur lokalen Zeit-zone (LPI 1: 107.3). Kann auch ein Link auf die passende Datei in



`/usr/share/zoneinfo/`

sein. Diese Datei ist sehr wichtig, denn sie bestimmt, wie die Uhrzeit von allen Anwendungen aus standardmäßig dargestellt wird.



Hinweis: Unter Debian kann man die Zeitzone mit

```
# dpkg-reconfigure tzdata
```


Lokalisierung und Internationalisierung

Ⓛ setzen. Alternativ gibt es dafür auch das Kommando **tzconfig** (LPI 1: 107.3), welches mittlerweile als veraltet gilt, jedoch für die LPI-Prüfung noch relevant ist (nicht unter Ubuntu).

Unter RedHat/Fedora kann man die Zeitzone in der Variablen `ZONE` in der Datei `/etc/sysconfig/clock` konfigurieren.

Zeitzone erforschen kann man mit dem Kommando **tzselect**: Über mehrere Menüs hinweg kann man eine bestimmte Zeitzone auswählen; es wird dann die Uhrzeit in der angegebene Zeitzone, sowie der Name der Zeitzone, wie er für die Umgebungsvariable `TZ` benötigt wird, ausgegeben.


Shell-Variable TZ Ein Benutzer kann durch Setzen der Shell-Variablen `TZ` die Einstellung aus `/etc/localtime` übergehen (um den richtigen Zeitzonennamen zu finden, verwende man das Kommando **tzselect**) (LPI 1: 107.3). Ⓛ

Beispiel: 

```
$ date
Do 26. Feb 15:27:33 WIT 2009
$ TZ='Pacific/Tongatapu'; export TZ
$ date
Do 26. Feb 21:27:37 TOT 2009
```

Durch Einfügen der genannten Zeile in `~/.bashrc` kann man die Konfiguration dauerhaft machen.

date: Die Systemzeit anzeigen und verändern (LPI 1: 107.3). Ⓛ

Beispiel: Systemdatum/-zeit setzen (flüchtig): 

```
# date --set 2010-12-24
# date --set 20:15:00
```

4.2 Zeichensätze, Encoding und iconv

Computer können nur Zahlen verarbeiten. Um auch Buchstaben in sinnvoller Weise verarbeiten zu können, benötigt man also eine Tabelle, die einem Zahlenwert eine Interpretation als Buchstabe zuordnet. Eine solche Tabelle nennt man *Encoding*.

Ⓔ **ASCII** war nach dem Morsecode der Nachfolger von Fernschreiber-Zeichensätzen wie Baudot und Murray. Der bis heute gültige Standard entstand 1968. Zu Beginn des Computer-Zeitalters wurde *ASCII* (LPI 1: 107.3) der Standard-Zeichensatz für viele Bildschirme und Drucker. Er enthält im Wesentlichen die Zeichen, die damals auf einer englischen Schreibmaschine zu finden waren und Steuerzeichen wie Zeilenvorschub oder Tabulator.

ASCII ist ein 7-Bit-Code, d.h. er definiert nur für die Zahlen 0-127 zugeordnete Zeichen (hexadezimal 00 bis 7F oder binär 0xxxxxxx, wobei x für 0 oder 1 steht).

ASCII hat auch heute noch eine zentrale Bedeutung, denn viele Programmiersprachen verwenden fast ausschließlich Zeichen, die in ASCII vorkommen. Wichtiger jedoch ist die Tatsache, dass fast alle heute eingesetzten Zeichensätze zu ASCII abwärtskompatibel sind, was bedeutet, dass sie ASCII als Teilmenge enthalten und man sie damit als eine Erweiterung von ASCII ansehen kann.

Die ASCII-Tabelle kann man sich unter Linux `man ascii` anzeigen lassen.

Die MS-DOS Codepages 437 und 852 Codepage 437 (englisch) war der acht Bit umfassende Zeichensatz des ersten IBM-PC und damit der Zeichensatz von MS-DOS. Im mitteleuropäischen Sprachraum wird dagegen die Codepage 852 eingesetzt. Sie unterscheidet sich von Codepage 437 nur in den Positionen 128-256 (7F bis FF hexadezimal) und enthält u.a. die deutschen Umlaute und andere akzentuierte Buchstaben.

Wegen der damals enorm starken Verbreitung von MS-DOS und Klonen des IBM-PC ist es durchaus wahrscheinlich, auch heute noch mit CP437 bzw. CP852-kodierten Texten in Kontakt zu kommen.

Ⓔ **ISO-8859** wurde 1998 von der ISO verabschiedet; *ISO-8859* (LPI 1: 107.3) ist eine 8-Bit-Erweiterung von ASCII und nutzt die zusätzlichen 127 Positionen, um Zeichen darzustellen, die in nicht-englischen Sprachen vorkommen. *ISO-8859* gliedert sich in 16 Substandards, welche mit *ISO-8859-1* bis *ISO-8859-16* bezeichnet werden.

Der wohl am häufigsten eingesetzte Substandard ist *ISO-8859-1* (Auch als *ISO-Latin-1* bekannt). Später wurde dieser um das Euro-Symbol und um einige andere in europäischen Sprachen fehlende Zeichen ergänzt und als *ISO-8859-15* definiert.

Linux baute in den neunziger Jahren fast vollständig auf ASCII und *ISO-8859-1* auf. Erst um die Jahrtausendwende herum wurde auf UTF-8 umgestellt.

ISO-8859-1 ist deshalb so bedeutsam, weil er als *die* De-Facto-Standard ASCII-Erweiterung angesehen werden kann und im Internet häufig als Zeichensatz von Web-Seiten und E-Mails eingesetzt wurde (und teilweise noch wird).

Außerdem sind die ersten 256 Zeichen des heutigen Standards Unicode mit *ISO-8859-1* identisch.

Lokalisierung und Internationalisierung

Details zu den ISO-8859-Standards kann man nachschlagen mit:

```
$ man iso_8859-1
$ man iso_8859-2
...
$ man iso_8859-15
```

Windows Codepage 1252 war / ist der Standard-Zeichensatz der Windows-Betriebssysteme. Er ist fast identisch mit ISO-8859-1, jedoch wurden einige Positionen modifiziert. Besonders störend wirkt sich hier aus, daß u.a. die Umlaute nicht kompatibel zu ISO-8869-1 sind. Obwohl er gerne als Windows-ANSI bezeichnet wird, wurde er niemals standardisiert (auch nicht von ANSI); er ist damit ein nicht standardkonformer Zeichensatz, der jedoch durch die Verbreitung des Microsoft Internet Explorer relativ häufig im Internet anzutreffen war.

Viele Windows-Texte sind heute noch 1252-kodiert abgespeichert, es ist also durchaus wahrscheinlich mit derart kodierten Textdateien in Berührung zu kommen

Unicode und UTF-8 *Unicode* (LPI 1: 107.3) stellt mit der Kodierung UTF-8 heute den Status Quo dar: Unicode hat das Ziel, alle heute verwendeten von Menschen benutzten (Schrift)-Zeichen in einer einzigen (sehr großen) Tabelle abzubilden. Ⓛ

Anfänglich wurde Unicode als 16-Bit-Kodierung konzipiert, was sich jedoch als zu eng herausstellte; daher ist Unicode heute in 17 Ebenen (Planes) zu je 16 Bit definiert:

Ebene 0 alias *Basic Multilingual Plane (BMP)*⁶, ist die wichtigste; sie enthält die Schriftzeichen aller lebenden Sprachen und der gebräuchlichen Sonderzeichen.

Ebene 1 alias *Supplementary Multilingual Plane (SMP)*⁷ umfasst historische Schriftsysteme, sowie seltener verwendete Symbole, wie z.B. Dominosteine.

Ebene 2 alias *Supplementary Ideographic Plane (SIP)*⁸ umfasst CJKV-Schriftzeichen (CJKV=Chinesisch, Japanisch, Koreanisch, Vietnamesisch), die seltener benutzt werden

Ebenen 3-13 sind noch nicht belegt

Ebene 14 alias *Special Purpose Plane (SSP)*

Unicode-Zeichen (*Codepoints* genannt) werden wie folgt als ASCII-Zeichen dargestellt:

⁶<http://unicode.org/roadmaps/bmp/>

⁷<http://unicode.org/roadmaps/smp/>

⁸<http://unicode.org/roadmaps/sip/>

Syntax:

`U+Hexadezimal-Zahl`

Die Hexadezimalzahl bewegt sich dabei im Bereich von 0000 bis 10FFFF, so dass U+0000 das erste definierte Unicode-Zeichen und U+10FFFF das letzte definierte Unicode-Zeichen darstellt.

Unicode hat den Vorzug, dass es von U+0000 bis U+007F (dezimal 0 bis 127) mit ASCII identisch ist und, von U+0080 bis U+00FF, mit ISO-8859-1.

Bei einer derart großen Tabelle von Schriftzeichen ergibt sich natürlich das Problem, diese möglichst speichereffizient abzulegen.

Es existiert daher eine Reihe von Kodierungen für Unicode; am gebräuchlichsten ist dabei das *Unicode Transformation Format (UTF)*, welches die Codepoints auf eine Folge von Bytes abbildet:

Ⓛ **UTF-8** ist die am weitesten verbreitete 8-Bit Kodierung (Internet, Desktop- und Serverbetriebssysteme), wobei ein Codepoint durch ein oder mehrere Bytes dargestellt wird. *UTF-8* (LPI 1: 107.3) zeichnet sich dadurch aus, dass es mit ASCII identisch ist, sofern nur Zeichen aus dem Bereich U+0000 bis U+007F verwendet werden. Für alle anderen Zeichen (inklusive der deutschen Umlaute und dem scharfen „S“) werden zwei Bytes (oder auch mehr) verwendet. Dabei werden umso mehr Bytes benötigt, je seltener ein Zeichen ist. Falls man also Text in ISO-8859-1 vorliegen hat, bedarf es eines Programms wie `iconv`, um die Umlaute richtig nach UTF-8 umzusetzen.

UTF-8 ist die platzsparendste Kodierung von Unicode, wenn es sich um Text handelt, der auf lateinischen Buchstaben basiert.

UTF-16 ist die interne Zeichenkodierung von Java und die Standard-Kodierung auf IBM-Mainframes. Es kommt jedoch relativ selten auf „freier Wildbahn“ vor und verwendet ein oder zwei 16-Bit-Wörter für die Darstellung eines Codepoints.

UTF-32 ist das einfachste UTF-Format: Es bildet jeden Codepoint der Ebene 0 (U+0000 bis U+FFFF) direkt (ohne variable Länge) ab. Es ist damit das einfachste UTF-Format, braucht jedoch für Texte, die auf dem lateinischen Alphabet basieren, etwa den vierfachen Speicherplatz.

Ⓛ **Textdateien konvertieren mit `iconv`** In heterogenen Arbeitsplatzumgebungen (Linux/Windows) oder wenn Bestandsdaten (z.B. alte E-Mails) mit aktuellen Systemen verarbeitet werden sollen, ist es notwendig, die Kodierung (Encoding) von Textdateien zu konvertieren. Diese Aufgabe erfüllt das Kommando `iconv` (LPI 1: 107.3):

Lokalisierung und Internationalisierung


Syntax:

```
iconv -l
iconv -f Quellkode -t Zielkode Quelldatei -o Zieldatei
iconv -f Quellkode -t Zielkode Quelldatei(en)
```

Die erste dargestellte Benutzungsvariante listet alle unterstützten Kodierungen auf.


Die zweite Variante konvertiert *eine* Quelldatei in *eine* Zieldatei.

Die dritte Variante kann *mehrere* Eingabedateien verarbeiten, gibt jedoch die Ergebnisse nacheinander auf der Standardausgabe aus.

Hinweis: Nicht alle von `iconv -l` angegebenen Zeichensätze lassen sich beliebig ineinander konvertieren. `iconv` gibt in diesem Fall eine Fehlermeldung aus. 

Beispiel:

```
$ cat > text-utf-8.txt
Dies ist deutscher Text: äöüÄÖÜß
Strg-D
$ iconv -f UTF-8 -t ASCII text-utf-8.txt
Dies ist deutscher Text: iconv: illegal input sequence at position 25
```



Wir erhalten hier eine Fehlermeldung, weil die Umlaute nicht in ASCII enthalten sind. Es gibt nun zwei Methoden, mit einem solchen Fall umzugehen: Die nicht vorhandenen Zeichen transliterieren oder ignorieren.

```
$ iconv -f UTF-8 -t ASCII//TRANSLIT text-utf-8.txt
Dies ist deutscher Text: aeoeueAEOEUEss
```

```
$ iconv -f UTF-8 -t ASCII//IGNORE text-utf-8.txt
Dies ist deutscher Text:
iconv: illegal input sequence at position 40
```

Im letzten Fall ist ein Fehler aufgetreten. Eigentlich sollte es `iconv` nicht schwerfallen die nichtübersetzbaren Zeichen zu ignorieren. Dies deutet auf einen Programmfehler (Bug) hin.

```
$ iconv -f UTF-8 -t CP852 text-utf-8.txt -o text-dos-cp852.txt
$ iconv -f UTF-8 -t WINDOWS-1252 text-utf-8.txt \
> -o text-windows-cp1252.txt
$ iconv -f UTF-8 -t ISO-8859-1 text-utf-8.txt \
> -o text-iso-8859-1.txt
```

Sehen Sie sich den Inhalt der konvertierten Dateien mit **cat** und einem Texteditor Ihrer Wahl an!

4.3 Sprache, Währungen, Datumsformate, ...

Da in jedem Sprachraum unterschiedliche Konventionen bezüglich Währungen, Datumsformaten, Papiergrößen, Telefonnummern und Maßeinheiten gelten, kann man auch diese Einstellungen per Umgebungsvariable setzen. Die aktuell wirksamen Einstellungen zeigt das Kommando **locale** (LPI 1: 107.3):



\$ locale	
LANG=de_DE.UTF-8	Voreinstellung, wird durch LC_* übergangen
LC_CTYPE="de_DE.UTF-8"	Zeichenklassifizierung und Groß/Kleinschreibung
LC_NUMERIC="de_DE.UTF-8"	Zahlendarstellung (nichtmonetär)
LC_TIME="de_DE.UTF-8"	Zeit- und Datenformate
LC_COLLATE="de_DE.UTF-8"	Sortierreihenfolge
LC_MONETARY="de_DE.UTF-8"	Zahlendarstellung (monetär)
LC_MESSAGES="de_DE.UTF-8"	Sprache der Textausgaben eines Programms
LC_PAPER="de_DE.UTF-8"	Papierformat
LC_NAME="de_DE.UTF-8"	Namensformat
LC_ADDRESS="de_DE.UTF-8"	Adress- und Ortsangaben
LC_TELEPHONE="de_DE.UTF-8"	Telefonnummern
LC_MEASUREMENT="de_DE.UTF-8"	Maßeinheiten
LC_IDENTIFICATION="de_DE.UTF-8"	Beschreibung der verwendeten Locale
LC_ALL=	Erzwingt die angegebene Locale für LC_*

Die Variable `LANG` wird man hier wohl am häufigsten benutzen: Sie bestimmt die Voreinstellung, falls die Variablen `LC_*` nicht gesetzt sind. Falls schon, so hat die Einstellungen der jeweiligen `LC_-`Variable Vorrang gegenüber `LANG`. Ist dagegen `LC_ALL` nicht leer, so hat diese Vorrang gegenüber allen anderen Einstellungen, mit anderen Worten: Es werden dann sowohl die Einstellungen von `LANG` als auch `LC_*` nicht mehr berücksichtigt.

Der Wert der Variablen muss mit dem Verzeichnisnamen eines Unterverzeichnisses von `/usr/lib/locale/` übereinstimmen. Diese Verzeichnisnamen sind nach folgender Konvention aufgebaut:

Konvention von Locale-Namen:

Sprachkürzel_Länderkürzel.Zeichensatz