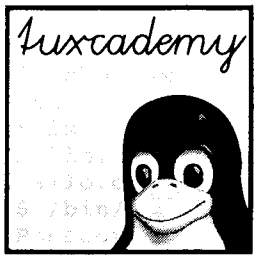

10

Lokalisierung und Internationalisierung

Inhalt

Lernziele

- Die gängigen Zeichencodierungen kennen
- Textdateien von einer Zeichencodierung in eine andere konvertieren können
- Die Umgebungsvariablen zur Spracheneinstellung kennen
- Die Linux-Infrastruktur zur Lokalisierung kennen
- Verstehen, wie Linux mit Zeitzonen umgeht

Vorkenntnisse

- Umgang mit Linux-Kommandos
- Umgang mit einem Texteditor

10.1 Überblick

»Internationalisierung« (engl. *internationalisation* oder kurz *i18n*, weil zwischen dem ersten und dem letzten Buchstaben des Worts 18 andere Buchstaben stehen) ist die Ausgestaltung eines Softwaresystems, so dass eine spätere Lokalisierung möglich ist. »Lokalisierung« (engl. *localisation* oder *l10n*) ist die Anpassung eines internationalisierten Systems an die Gepflogenheiten eines bestimmten Kulturkreises. Dazu gehört in erster Linie die Sprache für die Benutzungsoberfläche und allfällige Meldungen des Systems sowie die zu verarbeitenden Daten (die gegebenenfalls besondere Schriften und Eingabemethoden benötigt), aber auch Aspekte wie standardisierte Schreibweisen für Datums- und Zeitangaben, Geldbeträge, die Reihenfolge der Zeichen im Alphabet und einiges mehr. Bei grafischen Programmen können sogar bestimmte Farben der Lokalisierung unterliegen; in der westlichen Welt suggeriert zum Beispiel die Farbe Rot Gefahr, aber das ist nicht überall sonst so.

Für Linux als Betriebssystem-Kernel ist Internationalisierung nicht wirklich ein bedeutsames Thema, da der Kernel sich mit den meisten Problemfeldern, die einer Internationalisierung bedürfen, überhaupt nicht beschäftigt. (Man ist sich relativ einig, dass der Kernel zum Beispiel nicht mit Systemmeldungen in allen möglichen Sprachen überfrachtet werden sollte; von jemandem, der diese Meldungen überhaupt zu sehen bekommt, wird erwartet, dass er genug Englisch kann, um sie zu verstehen.) Linux-*Distributionen* dagegen enthalten riesige Mengen von Anwendersoftware, die von einer Lokalisierung profitieren dürfte. Entsprechend stehen die gängigen großen Distributionen auch in einer Vielzahl von lokalisierten Versionen zur Verfügung. Daneben gibt es diverse spezielle Linux-Distributionen, die sich auf einzelne Kulturkreise beschränken und diese besonders gut zu unterstützen versuchen.

Während große kommerzielle Softwarehersteller die Lokalisierung ihrer Software entweder von örtlichen Niederlassungen oder bezahlten Partnerfirmen vornehmen lassen, beruht die Lokalisierung von Open-Source-Software wie den gängigen Linux-Anwendungen in weiten Teilen auf der Mithilfe von Freiwilligen. Der Vorteil davon ist, dass sich in der Regel auch für sehr ausgefallene Sprachen Freiwillige finden lassen, die eine Software übersetzen und anpassen können – Sprachen, die zu unterstützen sich für einen kommerziellen Hersteller nie rechnen würde. Auf der anderen Seite stellt der Einsatz von Freiwilligen besondere Herausforderungen an die Qualitätskontrolle. Legendär ist die Anekdote, dass ein KDE-Entwickler aus der arabischen Welt bei der Lokalisierung alle Verweise auf den Staat Israel durch »Besetztes Palästina« ersetzt hat; ein Schritt, der bei der KDE-Gemeinde insgesamt nicht wirklich gut ankam …

10.2 Zeichencodierungen

Vorbedingung für die Internationalisierung und Lokalisierung von Programmen in fremden Sprachen (unter dem Strich »alles außer Englisch«) ist, dass das System den Zeichenvorrat der betreffenden Sprache anzeigen können muss. Der tra-

ASCII ditionelle Zeichencode für Computer ist der ASCII oder *American Standard Code for Information Interchange*, der, wie der Name sagt, für amerikanisches Englisch gedacht ist – in der Frühzeit elektronischer Rechenanlagen reichte das aus, aber alsbald wurde es notwendig, auch auf die Belange von »Fremdsprachen« Rücksicht zu nehmen.

Zeichenvorrat **ASCII** Der ASCII kann 128 verschiedene Zeichen darstellen, von denen 33 (die
Steuerzeichen Positionen 0–31 und die Position 127) für Steuerzeichen wie »Zeilenvorschub«, »Tabulator« oder »Glocke« reserviert sind. Die anderen 95 Zeichen umfassen

Tabelle 10.1: Die wichtigsten Teile von ISO/IEC 8859

Teil	Trivialname	Umfang
ISO 8859-1	Latin-1 (Westeuropa)	Die meisten westeuropäischen Sprachen: Dänisch, Deutsch, Englisch, Färöisch, Finnisch*, Französisch*, Isländisch, Irisch, Italienisch, Niederländisch*, Norwegisch, Portugiesisch, Rätoromanisch, schottisches Gälisch, Spanisch und Schwedisch. Daneben einige andere Sprachen, unter anderem Afrikaans, Albanisch, Indonesisch und Swahili. (* = teilweise)
ISO 8859-2	Latin-2 (Mitteleuropa)	Mittel- und osteuropäische Sprachen, die das lateinische Alphabet verwenden, etwa Bosnisch, Kroatisch, Polnisch, Serbisch, Slowakisch, Slowenisch, Tschechisch und Ungarisch.
ISO 8859-3	Latin-3 (Südeuropa)	Maltesisch, Türkisch und Esperanto.
ISO 8859-4	Latin-4 (Nordeuropa)	Estnisch, Lettisch, Litauisch, Grönländisch und Sami.
ISO 8859-5	Latin/Cyrillic	Die meisten slawischen Sprachen mit kyrillischem Alphabet, etwa Bulgarisch, Mazedonisch, Russisch, Serbisch, Ukrainisch (teilweise) und Weißrussisch.
ISO 8859-6	Latin/Arabic	Enthält die gängigsten arabischen Zeichen, ist aber nicht für Sprachen mit arabischer Schrift außer Arabisch geeignet. Arabisch wird von rechts nach links geschrieben!
ISO 8859-7	Latin/Greek	Modernes Griechisch und Altgriechisch (ohne Akzente)
ISO 8859-8	Latin/Hebrew	Modernes Hebräisch
ISO 8859-9	Latin-5 (Türkisch)	Entspricht im wesentlichen ISO 8859-1, mit türkischen Zeichen statt den isländischen. Auch für Kurdisch.
ISO 8859-10	Latin-6 (Nordisch)	Eine andere Anordnung von Latin-4.
ISO 8859-11	Latin/Thai	Für Thai.
ISO 8859-12	Latin/Devanagari	Wurde nie fertiggestellt.
ISO 8859-13	Latin-7 (Baltikum)	Enthält noch mehr Zeichen für baltische Sprachen, die in Latin-4 und Latin-6 nicht vorkommen.
ISO 8859-14	Latin-8 (Keltisch)	Keltische Sprachen wie Gälisch und Bretonisch.
ISO 8859-15	Latin-9	Im wesentlichen Latin-1, aber mit dem Euro-Zeichen und den für Estnisch, Finnisch und Französisch fehlenden Zeichen anstelle einiger fast nie gebrauchter anderer.
ISO 8859-16	Latin-10	Für Albanisch, Italienisch, Kroatisch, Polnisch, Rumänisch, Slowenisch und Ungarisch, aber auch Deutsch, Finnisch, Französisch und irisches Gälisch. Mit Euro-Zeichen. Ohne praktische Bedeutung.

Groß- und Kleinbuchstaben, Ziffern und eine Auswahl von gängigen Sonderzeichen, vor allem Interpunktion.

In Deutschland war die Zeichencodierung DIN 66003 gebräuchlich, die im wesentlichen dem ASCII entsprach, bis darauf, dass die eckigen und geschweiften Klammern, der vertikale Balken und der Rückstrich durch Umlaute, die Tilde durch das scharfe S und der Klammeraffe (»@«) durch das Paragraphenzeichen ersetzt wurde. Das war für Texte annehmbar, aber nicht notwendigerweise für C-Programme. (Der Autor dieser Zeilen erinnert sich noch an den ersten Drucker, mit dem er zu tun hatte, einen Centronics 737-2, der mit einem »Mäuseklavier«, also einer Bank von Mikroschaltern, von ASCII auf DIN 66003 und zurück umgeschaltet werden musste – ein mühseliges Unterfangen.)

DIN 66003

ISO/IEC 8859 Später stieg man unter dem zunehmenden Druck internationaler Computeranwender vom ASCII mit seinen 128 Zeichen auf erweiterte Zeichensätze um, die alle 256 möglichen Werte eines Bytes zur Zeichencodierung nutzten. Hervorzuheben ist hier der Standard ISO/IEC 8859, der Zeichencodierungen für viele verschiedene Sprachen vorsieht. Tatsächlich besteht ISO/IEC 8859 aus einer

256 Zeichen

Anzahl von numerierten, separat veröffentlichten Teilen, die auch als Standards angesehen werden können. Tabelle 10.1 gibt einen Überblick.

Informationsaustausch

Der Schwerpunkt der ISO/IEC-8859-Standards liegt auf Informationsaustausch, nicht eleganter Typografie, so dass diverse Zeichen, die für schöne Drucksachen erforderlich sind – etwa Ligaturen oder »typografische« Anführungszeichen – in den Zeichencodierungen fehlen. Bei der Zusammenstellung der Codetabellen konzentrierte man sich auf Zeichen, die bereits auf Computertastaturen vorhanden waren, und ging auch gewisse Kompromisse ein. Beispielsweise wurden die französischen »Œ«- und »œ«-Ligaturen nicht in Latin-1 aufgenommen, da man sie auch als »OE« und »oe« schreiben kann und der Platz in der Tabelle anderweitig benötigt wurde.

Grenzen

Fernöstliche Sprachen wie Chinesisch oder Japanisch werden von ISO/IEC 8859 nicht adressiert, da ihr Zeichenvorrat viel größer ist als die 256 Zeichen, die in eine einzelne ISO/IEC-8859-Codetabelle passen. Auch einige andere Alphabete der Welt sind nicht Gegenstand eines ISO/IEC-8859-Standards.

Unicode und ISO 10646 Unicode und ISO 10646 (das »Universal Character Set«) sind parallele Bemühungen, *alle* Alphabete der Welt in einer gemeinsamen Zeichencodierung zusammenzufassen. Die beiden Standards wurden zuerst getrennt entwickelt, aber dann – nachdem die Softwarefirmen der Welt heftigst gegen die Komplexität von ISO 10646 gemeutert hatten – zusammengelegt. Heutzutage standardisieren Unicode und ISO 10646 dieselben Zeichen mit denselben Zeichencodes; der Unterschied zwischen den beiden ist, dass ISO 10646 eine reine Zeichentabelle ist (im Prinzip ein erweitertes ISO 8859), während Unicode zusätzliche Regeln enthält, etwa für Sortierung, Normalisierung und bidirektionales Schreiben (für Sprachen wie Arabisch und Hebräisch). In diesem Sinne ist ISO 10646 eine »Untermenge« von Unicode; bei Unicode haben Zeichen noch diverse Zusatzeigenschaften, die zum Beispiel beschreiben, wie ein Zeichen sich mit anderen kombinieren läßt (was zum Beispiel für Arabisch wichtig ist, wo das Aussehen eines Zeichens davon abhängt, ob es am Anfang, in der Mitte oder am Ende eines Worts steht).

Zwei für alle

ISO 10646 ohne Unicode

Das Unix/Linux-Programm `xterm` ist ein Beispiel für ein Programm, das ISO 10646 unterstützt, aber nicht Unicode. Es kann alle ISO-10646-Zeichen darstellen, die sich direkt aus der Zeichencode-Tabelle ergeben und nur in eine Richtung geschrieben werden, und kann einige aus mehreren Zeichen (etwa einem »Basiszeichen« und Akzenten) zusammengesetzte Zeichen anzeigen, aber kein Hebräisch, Arabisch oder Devanagari. Unicode korrekt zu implementieren ist absolut kein Kinderspiel.

Zeichenvorrat

ISO 10646 enthält nicht nur Buchstaben und Ziffern, sondern auch Ideogramme (etwa chinesische und japanische Zeichen), mathematische Zeichen und vieles andere mehr. Jedes solche Zeichen wird durch einen eindeutigen Namen und eine Ganzzahl, seinen Codepunkt, identifiziert. Insgesamt stehen über 1,1 Millionen Codepunkte zur Verfügung, von denen allerdings nur die ersten 65.536 weithin verwendet werden. Diese nennt man auch die *basic multilingual plane* oder BMP. Unicode- bzw. ISO-10646-Codepunkte notiert man in der Form `U+0040`, wobei die vier Ziffern eine hexadezimale Zahl darstellen.

Codepunkte

BMP

UCS-2 und UTF-8 Unicode und ISO 10646 spezifizieren Codepunkte, also Nummern für die Zeichen im Code, aber geben zunächst nicht an, wie man mit diesen Codepunkten tatsächlich umgehen kann. Dafür sind sogenannte Codierungsformen definiert, die erklären, wie man die Codepunkte in einem Computer darstellt.

Codierungsformen

UCS-2

Die einfachste Codierungsform ist UCS-2, bei der für jedes Zeichen ein einzelner »Codewert« zwischen 0 und 65.535 verwendet wird, der in zwei Bytes dargestellt wird. Das heißt, UCS-2 ist beschränkt auf Zeichen in der BMP. UCS-2 hat auch das Problem, dass es für Daten aus der westlichen Welt, die in der Regel mit einer höchstens 8 Bit breiten Codierung wie ASCII oder ISO Latin-1 dargestellt

werden können, den doppelten Speicherplatz impliziert, da statt einem plötzlich zwei Byte pro Zeichen verbraucht werden.

Statt UCS-2 verwenden Systeme wie Windows inzwischen eine Codierungsform namens UTF-16, die es gestattet, auch Codepunkte außerhalb der BMP darzustellen. Das Speicherplatzproblem existiert trotzdem. UTF-16

UTF-8 ist in der Lage, jedes Zeichen in ISO 10646 darzustellen, aber ist dennoch rückwärtskompatibel mit ASCII und ISO-8859-1. Es codiert die Codepunkte `U+0000` bis `U+10FFFF` (also das 32fache von UCS-2) in ein bis vier Bytes, wobei die ASCII-Zeichen mit einem Byte auskommen. UTF-8 erfüllt die folgenden Entwurfsanforderungen: UTF-8

ASCII-Zeichen stehen für sich selbst Dadurch wird UTF-8 kompatibel zu allen Programmen, die mit Bytestrings umgehen können, also beliebigen Folgen von 8-Bit-Bytes, aber einigen ASCII-Zeichen eine Sonderbedeutung geben. So ist es leicht, existierende Systeme auf UTF-8 umzustellen.

Kein erstes Byte taucht in der Mitte eines Zeichens auf Wenn ein oder mehrere vollständige Bytes verlorengehen oder entstellt werden, dann ist es trotzdem möglich, den Anfang des nächsten Zeichens zu finden.

Das erste Byte eines Zeichens bestimmt die Byteanzahl Auf diese Weise wird sichergestellt, dass eine Bytefolge, die ein bestimmtes Zeichen darstellt, nicht Teil einer längeren Bytefolge sein kann, die für ein anderes Zeichen steht. Damit ist es effizient möglich, in Zeichenketten auf Byteebene nach Teilzeichenketten zu suchen.

Die Bytes `FE` und `FF` kommen nicht vor Diese Bytes stehen am Anfang eines UCS-2-Textes und dienen dazu, herauszufinden, in welcher Reihenfolge ein Rechner die beiden Bytes eines Worts darstellt. Da beide keine gültigen Bytes in einem UTF-8-Datenstrom sind, besteht so keine Verwechslungsgefahr zwischen UCS-2- und UTF-8-Daten.

UTF-8 ist inzwischen die Methode der Wahl für die Darstellung von Unicode-Daten auf einem Linux-System.

Wenn Sie genau wissen wollen, wie UTF-8 funktioniert, sollten Sie die Handbuchseite `utf-8(7)` konsultieren, die den Codierungsvorgang erklärt und mit Beispielen illustriert.

Das Kommando `iconv` erlaubt die Umwandlung zwischen verschiedenen Zeichencodierungen. Im einfachsten Fall konvertiert es den Inhalt einer auf der Kommandozeile benannten Datei (oder mehrerer Dateien) von einer angegebenen Zeichencodierung in die aktuell gültige. Das Resultat landet auf der Standardausgabe: iconv

```
$ iconv -f LATIN9 test.txt >test-utf8.txt
```

Sie können auch die gewünschte Zielcodierung angeben:

```
$ iconv -f UTF-8 -t LATIN9 test-utf8.txt >test-l9.txt
```

Mit `--output` (oder `-o`) können Sie eine Datei für die Ausgabe auswählen:

```
$ iconv -f LATIN9 -o test-utf8.txt test.txt
```

Ohne Eingabedateien liest `iconv` die Standardeingabe:

```
$ grep bla test.txt | iconv -f LATIN9 -o grep.out
```

Mit der Option `-l` können Sie sich ein Bild davon machen, welche Zeichencodierungen `iconv` kennt (was nicht notwendigerweise bedeutet, dass es erfolgreich zwischen beliebigen Paaren davon konvertieren kann).

Ungültige Zeichen

Wenn `iconv` in seiner Eingabe auf ein Zeichen stößt, das es nicht in die gewünschte Ausgabecodierung übertragen kann, dann meldet es einen Fehler und streicht die Segel. Als Abhilfe können Sie an die Zielcodierung eines der Suffixe `//TRANSLIT` oder `//IGNORE` anhängen. Mit `//IGNORE` werden alle in der Zielcodierung nicht vorhandenen Zeichen unter den Teppich gekehrt, während sie mit `//TRANSLIT`, wo möglich, durch ein oder mehrere ähnliche Zeichen approximiert werden:

```
$ echo xäöüy | iconv -f UTF-8 -t ASCII//IGNORE
xy
iconv: ungültige Eingabe-Sequenz an der Stelle 9
$ echo xäöüy | iconv -f UTF-8 -t ASCII//TRANSLIT
xaeoeuey
```

Die Option `-c` unterdrückt ungültige Zeichen kommentarlos:

```
$ echo xäöüy | iconv -c -f UTF-8 -t ASCII
xy
```

10.3 Spracheneinstellung unter Linux

Die Sprache, die ein Linux-System »spricht«, wird in der Regel bei der Installation aus einem komfortablen Menü ausgewählt und später nicht mehr geändert. Notfalls bieten die Arbeitsumgebungen KDE und GNOME ebenfalls eine komfortable Sprachauswahl an. Als Linux-Anwender kommen Sie also selten in die Verlegenheit, die Sprache explizit und über die Kommandozeile ändern zu müssen, aber auch das ist möglich.

Sprache: pro Sitzung

Als erstes müssen wir festhalten, dass die »Systemsprache« keineswegs eine Eigenschaft des kompletten Systems ist, sondern sich jeweils nur auf eine »Sitzung« bezieht. Normalerweise wird Ihre Login-Shell oder Ihre Arbeitsumgebung mit einer bestimmten Spracheinstellung initialisiert, und alle Prozesse, die von dieser abstammen, »erben« diese Einstellung so, wie zum Beispiel auch das aktuelle Verzeichnis oder die Ressourcenlimits von einem Prozess an seine Kindprozesse vererbt werden. Es spricht also nichts dagegen, dass Sie das System mit einer deutschen Spracheinstellung verwenden, während gleichzeitig jemand über das Netz oder an einem anderen Bildschirm angemeldet ist, der eine englische oder französische Spracheinstellung benutzt.

Genau genommen hält Sie auch niemand davon ab, dass Sie selbst in einem oder mehreren Fenstern eine andere Sprache einstellen als die Standardsprache Ihrer Sitzung.

LANG
Sprachencode
Ländercode

Maßgeblich für die Sprache einer Sitzung ist der Wert der Umgebungsvariablen `LANG`. Im einfachsten Fall besteht er aus einem Sprachencode gemäß ISO 639, gefolgt von einem Unterstrich, gefolgt von einem Ländercode gemäß ISO 3166, also zum Beispiel etwas wie

`de_DE`	*Deutsch in Deutschland*
`de_AT`	*Deutsch in Österreich*

Die Länderunterscheidung ist wichtig, da die Sprachen zweier Länder sich durchaus unterscheiden können, selbst wenn sie eigentlich dieselbe Sprache verwenden. Unsere österreichischen Freunde sagen bekanntlich »Jänner« statt »Januar«

und geben ihren Gemüsen merkwürdige Namen wie »Paradeiser« und »Karfiol« – ersteres ist wichtig, wenn Ihr Linux-System das Datum ausgeben soll, und bei letzterem ist es zumindest denkbar, dass ein Textverarbeitungsprogramm das Wort »Karfiol« in einem de_DE-Text genauso als merkwürdig anmeckert wie das Wort »Blumenkohl« unter de_AT.

Wenn der Unterschied zwischen de_DE und de_AT Ihnen nicht krass genug ist, dann denken Sie mal über en_GB und en_US nach, oder über pt_PT und pt_BR.

Hinter dieser einfachen Spezifikation können noch Erweiterungen folgen, etwa eine Zeichencodierung (hinter einem Punkt) oder eine »Variante« (hinter einem @). Sie können also Werte verwenden wie

Erweiterungen

de_DE.ISO-8859-15	*Deutsch in Deutschland, gemäß ISO Latin-9*
de_AT.UTF-8	*Deutsch in Österreich, basierend auf Unicode*
de_DE@euro	*Deutsch in Deutschland, mit Euro (ISO Latin-9)*

Hier sehen Sie die Auswirkungen einer LANG-Änderung:

```
$ for i in de_DE de_AT en_US fi_FI fr_FR; do
>   LANG=$i.UTF-8 date +"%B %Y"
> done
Januar 2009
Jänner 2009
January 2009
tammikuu 2009
janvier 2009
```

(Bei date steht der Formatschlüssel %B für »Monatsname gemäß der aktuellen Spracheinstellung«.)

Das Ganze setzt natürlich voraus, dass es auf dem betreffenden System überhaupt Unterstützung für die jeweilige Sprache *gibt*. Debian GNU/Linux zum Beispiel läßt Ihnen die Wahl, welche Einstellungen gültig sein sollen oder nicht. Wenn Sie sich eine Einstellung wünschen, die Ihr System nicht unterstützt, dann fällt das System auf einen eingebauten Standardwert zurück, und das ist in der Regel Englisch.

Die Umgebungsvariable LANGUAGE (nicht zu verwechseln mit LANG) wird nur von Programmen beachtet, die die GNU-gettext-Infrastruktur verwenden, um ihre Meldungen in verschiedene Sprachen zu übersetzen (und das sind unter Linux die meisten). Der offensichtlichste Unterschied zwischen LANGUAGE und LANG ist, dass LANGUAGE es erlaubt, mehrere Sprachen aufzuzählen (durch Doppelpunkte getrennt). Damit können Sie eine Präferenzliste angeben:

LANGUAGE

```
LANGUAGE=de_DE.UTF-8:en_US.UTF-8:fr_FR.UTF-8
```

bedeutet »Deutsch, oder sonst Englisch, oder sonst Französisch«. Die erste Sprache, für die ein Programm Systemmeldungen mitbringt, gewinnt. LANGUAGE hat (für Programme, die GNU gettext benutzen) Vorrang gegenüber LANG.

Übungen

10.1 [1] Was erscheint als Ausgabe des Kommandos

```
$ LANG=ja_JP.UTF-8 date +"%B %Y"
```

(installierte Unterstützung für diese Sprache vorausgesetzt)?

10.4 Lokalisierungs-Einstellungen

Tatsächlich beeinflusst der Wert von LANG nicht nur die Sprache, sondern die komplette »kulturelle Einstellung« eines Linux-Systems. Dazu gehören außerdem Sachen wie

Formatierung von Datums- und Zeitangaben In den USA ist es zum Beispiel üblich, ein Datum in der Form »Monat/Tag/Jahr« anzugeben:

```
$ date +"%x"                                    Landesabhängige Zeitangabe
14.01.2009
$ LANG=en_US.UTF-8 date +"%x"
01/14/2009
```

Ferner benutzen manche Länder (wieder die USA oder auch Großbritannien) eine 12-Stunden-Zeit, während zum Beispiel in Deutschland eine 24-Stunden-Zeit üblich ist: Was wir »15 Uhr« nennen würden, ist dort »3 p. m.«.

Formatierung von Zahlen und Geldbeträgen In Kontinentaleuropa verwenden wir ein Komma als Dezimaltrenner und Punkte, um die Lesbarkeit einer großen Zahl zu erhöhen:

```
299.792.458,0
```

In den USA dagegen ist es gerade umgekehrt:

```
299,792,458.0
```

Diese Einstellung wirkt sich zunächst auf Programme aus, die die C-Funktionen `printf()` und `scanf()` verwenden. Andere Programme müssen die Einstellung explizit abfragen und in ihrer Ausgabe berücksichtigen.

Bei Geldbeträgen kommt dazu, dass zum Beispiel negative Salden manchmal durch ein vorgesetztes Minus und manchmal durch Klammern kenntlich gemacht werden (unter anderem).

Klassifizierung von Zeichen Es hängt von der verwendeten Sprache ab, welche Zeichen in der betreffenden Codetabelle als Buchstaben gelten, welche als Sonderzeichen und so weiter. Im Zeitalter von Unicode ist das nicht mehr so ein Problem, da die Codepunkte insgesamt standardisiert sind; komplizierter ist es, wenn zum Beispiel ISO-8859- oder gar ASCII-basierte Codierungen verwendet werden. In ASCII ist »[« zum Beispiel ziemlich eindeutig ein Sonderzeichen, das »Ä«, das in DIN 66003 denselben Platz in der Tabelle belegt, aber ebenso eindeutig ein Buchstabe. Dies hat auch Auswirkungen auf die Konvertierung zwischen Groß- und Kleinbuchstaben und ähnliches.

Das deutsche scharfe S (»ß«) hat keine grafische Entsprechung als Großbuchstabe – ein Wort wie »Fuß« wird in Versalien »FUSS« geschrieben. (Bei Verwechslungsgefahr wurde sogar »SZ« empfohlen, etwa bei »MASSE« vs. »MASZE«, aber seit der neuen Rechtschreibung ist das abgeschafft.) Im Zusatz 4:2008 zu ISO 10646, der am 23. Juni 2008 veröffentlicht wurde, ist ein Codepunkt für ein »großes ß« (U+1E9E) enthalten, so dass grundsätzlich einer Behebung dieses Problems an der Wurzel nichts mehr im Weg steht. Wir müssen uns nur noch darüber einigen, wie dieses Zeichen tatsächlich aussehen soll. (»SS« wäre eine naheliegende Wahl.)

Tabelle 10.2: LC_*-Umgebungsvariable

Variable	Bedeutung
LC_ADDRESS	Formatierung von Adressen und Ortsangaben
LC_COLLATE	Zeichensortierung
LC_CTYPE	Zeichenklassifizierung und Groß- und Kleinschreibung
LC_MONETARY	Formatierung von Geldbeträgen
LC_MEASUREMENT	Maßeinheiten (metrisch oder anders)
LC_MESSAGES	Sprache für Meldungen und Gestalt von positiven bzw. negativen Antworten
LC_NAME	Formatierung von Eigennamen
LC_NUMERIC	Formatierung von Zahlen
LC_PAPER	Papierformat (umstritten)
LC_TELEPHONE	Formatierung von Telefonnummern
LC_TIME	Formatierung von Datums- und Zeitangaben
LC_ALL	Alle Einstellungen

Sortierreihenfolge von Zeichen Auch das ist keineswegs so eindeutig, wie man glauben mag. Schon in Deutschland gibt es zwei konkurrierende Methoden gemäß DIN 5007: In Lexika werden Umlaute als äquivalent zu ihren »Basiszeichen« angesehen (ein »ä« wird also für Zwecke der Sortierung als »a« interpretiert), während in Namenslisten, zum Beispiel Telefonbüchern, Umlaute gemäß ihrer »Ersatzschreibweise« sortiert werden (»ä« also wie »ae«). In beiden Fällen ist »ß« äquivalent zu »ss«.

Bei Namenslisten wünscht man sich offenbar, dass der Unterschied zwischen den Homophonen »Müller« und »Mueller« die Suche nicht zu sehr erschwert. Herrn Müller müßten Sie sonst zwischen Frau Muktadir und Herrn Muminovic suchen, während Frau Mueller zwischen Herrn Muders und Frau Muffert zu finden wäre – eine Unbequemlichkeit erster Güte. Im Lexikon dagegen sollte die Rechtschreibung des Suchbegriffs und damit dessen Einsortierung klar sein.

In Schweden dagegen stehen die Buchstaben »å«, »ä« und »ö« in dieser Reihenfolge am Ende des Alphabets (hinter »z«). In Großbritannien kommt »ä« nach »a« (also zwischen »az« und »b«), und gemeinerweise ist der Namensbestandteil »Mc« äquivalent zu »Mac« (die Sortierung ist also »Macbeth, McDonald, MacKenzie«). Sprachen auf der Basis von Ideogrammen wie Japanisch und Chinesisch sind noch unbequemer zu sortieren; in Wörterbüchern orientiert man sich an der Struktur der Ideogramme und ihrer Strichanzahl, während auf Computern bequemerweise nach der lateinischen Umschrift sortiert wird. (Wir hören an dieser Stelle auf, bevor Ihnen der Kopf platzt.)

Die LANG-Variable setzt neben der Sprache auch das alles mit einem Schlag auf die in einem bestimmten Kulturkreis (engl. *locale*) gültigen Werte. Daneben existiert aber auch die Möglichkeit, einzelne Aspekte der Lokalisierung separat einzustellen. Dafür gibt es eine Reihe von Umgebungsvariablen der Form LC_* (siehe Tabelle 10.2).

Wenn Sie wissen wollen, was diese Einstellungen tatsächlich beinhalten, können Sie das Kommando `locale` heranziehen, etwa so:

```
$ locale -k LC_PAPER
height=297
width=210
paper-codeset="UTF-8"
```

Auf diese Weise können Sie also herausfinden, dass Papierblätter in Deutschland in der Regel 297 mm hoch und 210 mm breit sind. Wir kennen das als »DIN A4«.

Die tatsächlichen *Definitionen*, die den Einstellungen zugrunde liegen, finden Sie im Verzeichnis `/usr/share/i18n/locales`. Grundsätzlich steht Ihnen nichts im Weg, wenn Sie eine eigene Einstellungsdatei entwerfen wollen (außer vielleicht der spärlichen Dokumentation). Das Programm `localedef` macht die eigentliche Arbeit.

`LC_ALL` Mit `LC_ALL` können Sie ähnlich wie mit `LANG` auf einen Schlag alle Kulturkreis-Einstellungen setzen. Bei der Bestimmung, welche Einstellung nun letzten Endes maßgeblich ist, geht das System wie folgt vor:

1. Wenn `LANG` gesetzt ist, dann gilt dessen Wert.
2. Wenn `LANG` nicht gesetzt ist, aber die `LC_*`-Variable für den betreffenden Bereich (etwa `LC_COLLATE`) gesetzt ist, dann gilt deren Wert.
3. Wenn weder `LANG` noch die passende `LC_*`-Variable gesetzt sind, aber die Variable `LC_ALL` gesetzt ist, dann gilt deren Wert.
4. Wenn überhaupt nichts dergleichen gesetzt ist, dann gilt ein fest ins Programm eingebauter Standardwert.

Beachten Sie also, dass Sie, wenn (wie üblich) die `LANG`-Variable gesetzt ist, mit den `LC_*`-Variablen machen können, was Sie wollen – ohne dass es irgendwelche Auswirkungen zeitigt.

Wenn Sie sich jetzt am Kopf kratzen und sich wundern, dann haben Sie völlig recht – warum sollte man überhaupt `LC_*`-Variable setzen, wenn `LANG`, die Umgebungsvariable, die das System beim Anmelden für Sie setzt, eh alles andere platt macht? Für uns ist das genauso ein Rätsel wie für Sie, aber es gibt ja notfalls `.bash_profile` und »`unset LANG`«.

Das Kommando »`locale -a`« liefert eine Liste von Werten, die das aktuelle System für `LANG` und die `LC_*`-Variablen unterstützt:

```
$ locale -a
C
de_AT.utf8
de_DE
de_DE@euro
de_DE.utf8
deutsch
◁◁◁◁◁
POSIX
```

Sachen wie `LANG=deutsch` sehen auf den ersten Blick verlockend aus, sind aber zu unkonkret, um nützlich zu sein. Offiziell verpönt sind sie außerdem, werden aber aus Kompatibilitätsgründen (noch) beibehalten. Machen Sie einen Bogen darum.

`C` `POSIX` Die magischen Werte `C` und `POSIX` (äquivalent) in der Liste beschreiben den fest eingebauten Standard, den Programme verwenden, wenn sie keine gültige andere Einstellung finden. Dies ist nützlich, wenn Sie möchten, dass Programme wie `ls` ein definiertes Ausgabeformat liefern. Vergleichen Sie zum Beispiel

```
$ LANG=de_DE.UTF-8 ls -l /bin/ls
-rwxr-xr-x 1 root root 92312  4. Apr 2008  /bin/ls
$ LANG=ja_JP.UTF-8 ls -l /bin/ls
```

```
-rwxr-xr-x 1 root root 92312 2008-04-04 16:22 /bin/ls
$ LANG=fi_FI.UTF-8 ls -l /bin/ls
-rwxr-xr-x 1 root root 92312  4.4.2008 /bin/ls
$ LANG=C ls -l /bin/ls
-rwxr-xr-x 1 root root 92312 Apr  4  2008 /bin/ls
```

Wir führen dasselbe Kommando mit vier verschiedenen LANG-Einstellungen aus und erhalten vier verschiedene Resultate, die sich alle in der Datumsangabe unterscheiden. Gemeinerweise kann diese Datumsangabe, je nach Spracheinstellung, aus der Sicht von Programmen wie awk oder »cut -d' '« aus einem, zwei oder drei Feldern bestehen – fatal, wenn ein Skript diese Ausgabe analysieren soll! Sie tun also gut daran, in solchen Zweifelsfällen Programme wie ls, deren Ausgabeformat von der Sprache abhängt, per explizitem LANG=C auf einen Standard festzunageln, der auf jeden Fall existiert (bei allen anderen Einstellungen können Sie sich da nicht sicher sein).

Übungen

10.2 [2] Das Programm printf(1) (nicht zu verwechseln mit dem eingebauten Shellkommando printf) richtet sich beim Formatieren von Zahlen nach der LC_NUMERIC-Einstellung. Insbesondere formatiert ein Kommando wie

```
$ /usr/bin/printf "%'g\n" 31415.92
```

eine Dezimalzahl mit dem für die aktuelle Spracheinstellung korrekten Dezimaltrenner und »Übersichtlichkeitszeichen«. Experimentieren Sie mit dem Programm und verschiedenen Einstellungen für LANG.

10.3 [2] Finden Sie Programme außer date, ls und printf, die sich nach Sprach- bzw. Kulturkreiseinstellungen richten. (Übersetzte Fehlermeldungen sind zu einfach und zählen nicht, es muss schon etwas Interessantes sein.)

10.5 Zeitzonen

Zu guter Letzt müssen wir noch ein paar Dinge über Zeitzonen sagen und beschreiben, wie Linux mit ihnen umgeht. Spätestens wenn Sie mal eine weite Flugreise nach Westen oder Osten unternommen haben, dürfte Ihnen klar sein, dass es nicht überall auf der Erde immer gleich spät ist – steht bei uns in Europa die Sonne hoch am Himmel, ist es in Amerika möglicherweise noch dunkel und in Ostasien neigt der Tag sich schon wieder dem Ende zu. Das wäre alles kein Problem (Ihren Wecker stellen Sie ja auch nach dem Zeitsignal im Radio), wenn da nicht das Internet wäre, das es ermöglicht, Daten mit hoher Geschwindigkeit zwischen irgendwelchen Computern irgendwo auf der Welt auszutauschen. Und ob eine E-Mail nun um 12 Uhr europäischer, amerikanischer oder australischer Zeit geschrieben wurde, ist schon ein Unterschied von etlichen Stunden, auf den man Rücksicht nehmen möchte.

Heutige Computersysteme gestatten es also, festzulegen, in welcher Zeitzone der Rechner aufgestellt ist – typischerweise werden Sie bei der Systeminstallation danach gefragt, und wenn Sie nicht gerade mit dem Rechner im Gepäck auswandern, müssen Sie den einmal gesetzten Wert auch nie wieder ändern.

Zeitzone

Die Zeitzonen der Erde richten sich im wesentlichen nach dem Umstand, dass ein Unterschied von 15 Grad in geografischer Länge einem Unterschied von einer Stunde auf der Uhr entspricht (was logisch ist, denn der komplette Erdumfang im Werte von 24 Stunden entspricht ja 360 Grad, und 360/24

Tabelle 10.3: Die Sommerzeit in Deutschland

Zeitraum	Situation
vor 1916	Keine Zeitumstellung
1916	Sommerzeit vom 1. Mai bis zum 1. Oktober
1917–1918	Sommerzeit von Mitte April bis Mitte September
1919–1939	Keine Zeitumstellung
1940–1942	Die Uhr wurde am 1. April 1940 um eine Stunde vorgestellt und blieb so bis zum 2. November 1942 (!)
1943–1944	Sommerzeit von Ende März/Anfang April bis Anfang Oktober
1945	Sommerzeit vom 2. April bis 18. November, außerdem zwischen dem 24. Mai und dem 24. September »doppelte« Sommerzeit (die Uhr wurde nochmals eine Stunde vorgestellt)
1946–1949	Sommerzeit von Mitte April bis Anfang Oktober
1947	»Doppelte« Sommerzeit zwischen dem 11. Mai und dem 29. Juni
1950–1979	Keine Zeitumstellung
1980–1995	Sommerzeit vom letzten März- bis zum letzten Septembersonntag (war in der Bundesrepublik schon 1978 beschlossen worden, aber man musste sich mit der DDR einigen)
seit 1996	Sommerzeit vom letzten März- bis zum letzten Oktobersonntag (EU-Vereinheitlichung)

ist nun mal 15). Früher gab es keine Zeitzonen, sondern jeder Ort hatte seine eigene Zeit, wobei man Wert darauf legte, dass die Sonne um 12 Uhr mittags möglichst genau im Süden stehen sollte (vermutlich damit die Sonnenuhren stimmen). Die Einführung des Eisenbahnverkehrs und die mit der »Ortszeit« verbundenen Schwierigkeiten bei der Fahrplan-Erstellung machten das jedoch zusehends unpraktisch, so dass man sich zur Einführung der Zeitzonen entschloss, um den Preis, dass die Zeit auf der Uhr lokal nicht mehr mit der »astronomischen« Zeit übereinstimmt. Die Grenzen der Zeitzonen orientieren sich ohnehin nicht (ausschließlich) an Längengraden, sondern eher an politischen Grenzen.

Dass das in der Praxis durchaus merkbar sein kann, sieht man in Europa. Spanien zum Beispiel folgt der »mitteleuropäischen Zeit« (MEZ), die auf 15 Grad östlicher Länge bezogen ist. Das entspricht zum Beispiel Görlitz an der deutsch-polnischen Grenze. Wenn es in A Coruña an der spanischen Atlantikküste (fast 8,5 Grad *westlicher* Länge) auf der Uhr 12 Uhr mittags ist, dann ist die Sonne erst auf ungefähr halb elf. (Portugal verwendet übrigens dieselbe Zeit wie Großbritannien; dort ist es dann immerhin schon fast halb zwölf.)

Weiter verkompliziert wird das Ganze durch die »Sommerzeit« (engl. *daylight savings time*), bei der die Uhren in einer Zeitzone im Frühjahr künstlich eine Stunde vorgestellt werden und im Herbst wieder zurück. Die Sommerzeit ist ein rein politisches Phänomen, das aber trotzdem beachtet werden muss – in Deutschland hat sie eine durchaus bewegte Geschichte (Tabelle 10.3), die illustriert, warum Sie bei Linux nicht einfach »MEZ« als Zeitzone einstellen können, sondern sich »Europe/Berlin« wünschen müssen.

Wie bei den Sprach- und Kultureinstellungen gilt, dass die Zeitzone auf einem Linux-System keine eindeutige systemweite Einstellung ist, sondern zu den vererbbaren Eigenschaften eines Prozesses gehört. Der Linux-Kernel misst die Zeit in Sekunden seit dem 1.1.1970, 0 Uhr UTC, so dass der Komplex »Zeitzone« lediglich eine Frage der Formatierung dieser (inzwischen recht großen) Sekunden-

zahl[1]. Damit werden elegant alle Schwierigkeiten umgangen, die andere Betriebssysteme mit der Sommerzeit-Umstellung hatten und haben, und es besteht auch gar kein Problem darin, dass Ihr Kumpel aus Sydney sich über das Internet auf Ihrem Rechner anmeldet und die australische Zeit sieht, während Sie selbst natürlich MEZ verwenden. So gehört sich das.

Die Voreinstellung für die Zeitzone, die Sie bei der Installation des Systems machen, hinterlegt Linux in der Datei `/etc/timezone`:

```
$ cat /etc/timezone
Europe/Berlin
```

Die gültigen Zeitzonen können Sie den Namen der Dateien unter `/usr/share/zoneinfo` entnehmen:

```
$ ls /usr/share/zoneinfo
Africa/      Chile/   Factory    Iceland      MET       Portugal   Turkey
America/     CST6CDT  GB         Indian/      Mexico/   posix/     UCT
Antarctica/  Cuba     GB-Eire    Iran         Mideast/  posixrules Universal
Arctic/      EET      GMT        iso3166.tab  MST       PRC        US/
Asia/        Egypt    GMT0       Israel       MST7MDT   PST8PDT    UTC
Atlantic/    Eire     GMT-0      Jamaica      Navajo    right/     WET
Australia/   EST      GMT+0      Japan        NZ        ROC        W-SU
Brazil/      EST5EDT  Greenwich  Kwajalein    NZ-CHAT   ROK        zone.tab
Canada/      Etc/     Hongkong   Libya        Pacific/  Singapore  Zulu
CET          Europe/  HST        localtime@   Poland    SystemV/
```

Hierbei handelt es sich größtenteils um Verzeichnisse:

```
$ ls /usr/share/zoneinfo/Europe
Amsterdam  Chisinau     Kiev       Moscow     Sarajevo   Vatican
Andorra    Copenhagen   Lisbon     Nicosia    Simferopol Vienna
Athens     Dublin       Ljubljana  Oslo       Skopje     Vilnius
Belfast    Gibraltar    London     Paris      Sofia      Volgograd
Belgrade   Guernsey     Luxembourg Podgorica  Stockholm  Warsaw
Berlin     Helsinki     Madrid     Prague     Tallinn    Zagreb
Bratislava Isle_of_Man  Malta      Riga       Tirane     Zaporozhye
Brussels   Istanbul     Mariehamn  Rome       Tiraspol   Zurich
Bucharest  Jersey       Minsk      Samara     Uzhgorod
Budapest   Kaliningrad  Monaco     San_Marino Vaduz
```

Die Regel ist, dass die Zeitzone nicht etwa heißt wie die Hauptstadt des betreffenden Landes (das wäre zu einfach), sondern so wie die bevölkerungsreichste Stadt in dem Teil des betreffenden Landes, für den die Zeitzone gilt. Die Schweiz wird also durch `Europe/Zurich` abgedeckt (und nicht `Europe/Berne`), und Russland hat insgesamt 11 Zeitzonen, die allerdings nicht alle unter `Europe` gezählt werden. `Europe/Kaliningrad` ist zum Beispiel eine Stunde vor `Europe/Moscow`, und das wiederum ist wiederum zwei Stunden vor `Asia/Yekaterinburg`.

`/usr/share/zoneinfo` enthält auch einige »Bequemlichkeits-Zeitzonen« wie `Poland` oder `Hongkong`. `Zulu` hat nichts mit Südafrika zu tun, sondern bezieht sich, wie Leser von Tom Clancy wissen, auf die Weltzeit (UTC), die gerne als `12:00Z` angegeben wird, wobei die NATO »Z« als »Zulu« buchstabiert.

`/etc/localtime` ist eine Kopie der Datei unter `/usr/share/zoneinfo`, die die Informationen für die Zeitzone enthält, die durch den Inhalt von `/etc/timezone` gegeben ist – zum Beispiel `/usr/share/zoneinfo/Europe/Berlin`.

[1] Am 14. Februar 2009 um 0:31:30 Uhr MEZ sind seit dem Anbeginn der Linux-Zeitrechnung genau 1234567890 Sekunden vergangen. Fröhlichen Valentinstag!

Eigentlich sollte `/usr/share/zoneinfo` Zeitzonen für jeden Geschmack zur Verfügung stellen. Sollten Sie jemals in die Verlegenheit kommen, selber eine neue Zeitzone definieren zu müssen, dann können Sie das über den »Zeitzonen-Compiler« `zic` tun. Mit `zdump` können Sie die Zeit in jeder beliebigen Zeitzone oder auch die »Vorgeschichte« der Sommerzeit in jeder Zeitzone abrufen. (Raten Sie mal, wo wir die Informationen für Tabelle 10.3 her haben.)

Sie können die systemweite Zeitzone »manuell« setzen, indem Sie die Dateien `/etc/timezone` und `/etc/localtime` anpassen:

```
# echo Asia/Tokyo >/etc/timezone
# cp /usr/share/zoneinfo/$(cat /etc/timezone) /etc/localtime
```

Darüber hinaus bieten die Distributionen verschiedene Hilfsmittel zur komfortableren Konfiguration der Zeitzone an.

Mit `tzselect` können Sie interaktiv eine Zeitzone auswählen. Das Programm fragt Sie nach Erdteil und Land und bietet Ihnen dann eine Auswahl der gültigen Zeitzonen an. Es liefert den Namen der Zeitzone auf der Standardausgabe, während die komplette Interaktion sich auf der Standardfehlerausgabe abspielt; Sie können also mit etwas wie

```
$ TZ=$(tzselect)
```

das Ergebnis in eine Umgebungsvariable holen. – Zur Änderung der systemweiten Zeitzone wird nicht `tzselect` verwendet, sondern der `debconf`-Mechanismus; rufen Sie

```
# dpkg-reconfigure tzdata
```

auf. Das hin und wieder noch erwähnte Programm `tzconfig` ist verpönt.

SUSE-Anwender können die systemweite Zeitzone über den YaST setzen. Für die »persönliche« Zeitzone gibt es nichts offensichtliches Bequemes.

Die systemweite Voreinstellung für die Zeitzone steht in der Datei `/etc/sysconfig/clock`. Außerdem gibt es ein Programm namens `timeconfig`, und das bei Debian GNU/Linux diskutierte `tzselect` steht ebenfalls zur Verfügung.

Unabhängig von der systemweiten Voreinstellung für die Zeitzone können Sie in der Umgebungsvariablen `TZ` den Namen einer Zeitzone hinterlegen, die dann für den betreffenden Prozess verwendet wird (und sich wie andere Umgebungsvariablen an Kindprozesse vererbt). Mit etwas wie

```
$ export TZ=America/New_York
```

können Sie zum Beispiel für eine Shell und alle Programme, die Sie aus ihr heraus starten, die Zeitzone `America/New_York` setzen.

Sie können die Zeitzone auch nur für ein einziges Kommando ändern:

```
$ TZ=America/New_York date
```

zeigt Ihnen die aktuelle Zeit in New York.

Mit der Variablen `TZ` können Sie eine Zeitzone auch beschreiben, ohne auf die Zeitzonen-Daten in `/usr/share/zoneinfo` zurückgreifen zu müssen. Im einfachsten Fall geben Sie dazu einfach den (abgekürzten) Namen der Zeitzone und die Differenz zur UTC an. Letztere muss ein Vorzeichen haben (»+« für Zeitzonen westlich vom Nullmeridian, »-« für östlich), gefolgt von einer Zeitspanne der Form `HH:MM:SS` (der Minuten- und Sekundenanteil ist optional, und aktuell gibt es auch keine Zeitzonen mit einer Sekundenverschiebung). Etwas wie

```
$ export TZ=MEZ-1
```

würde also »mitteleuropäische Zeit« setzen, allerdings ohne Sommerzeit oder gar die deutsche Sommerzeit-Vorgeschichte. Um die Sommerzeit mit anzugeben, müssen Sie deren Namen, eine eventuelle Differenz zur Normalzeit (wenn sie nicht auf »eine Stunde vorgestellt« hinausläuft) und eine Regel für die Umstellung angeben. Die Umstellungsregel besteht aus einer Tagesangabe gefolgt von einer optionalen Zeitangabe (abgetrennt durch einen Schrägstrich), wobei die Tagesangabe eine von drei Formen haben kann:

`J`*n* Die Nummer des Tags im Jahr, gezählt von 1 bis 365. Der 29. Februar, wenn es ihn gibt, zählt nicht mit.

n Die Nummer des Tags im Jahr, gezählt von 0 bis 365. In Schaltjahren zählt der 29. Februar mit.

`M`*m*`.`*w*`.`*d* Tag *d* der Woche *w* in Monat *m*. *d* ist zwischen 0 (Sonntag) und 6 (Samstag), die Woche ist zwischen 1 und 5, wobei 1 für die erste und 5 für die letzte Woche mit einem Tag *d* steht (letzteres unabhängig davon, wie viele Tage *d* es im Monat tatsächlich gibt). *m* ist zwischen 1 und 12.

Die aktuell in Deutschland übliche Regel könnten Sie also mit

```
$ export TZ=MEZ-1MESZ,M3.5.0/2,M10.5.0/2
```

einstellen, wobei Ihnen aber auch hier die »Vorgeschichte« aus Tabelle 10.3 entgeht.

Übungen

10.4 [1] Warum ist `/etc/localtime` eine Kopie der relevanten Datei aus `/usr/share/zoneinfo`? Ein symbolisches Link würde doch auch reichen, oder man konsultiert direkt die richtige Datei in `/usr/share/zoneinfo`. Was denken Sie?

10.5 [!2] Stellen Sie sich vor, Sie sind Börsenmakler und brauchen einen schnellen Überblick über die Zeit in Tokyo, Frankfurt und New York. Wie können Sie das mit Linux verwirklichen?

10.6 [2] Geben Sie eine `TZ`-Umstellungsregel für das hypothetische Land Nirgendwonien an. Es gelten die folgenden Grundregeln:

- In Nirgendwonien gilt die Nirgendwonische Normalzeit (NNZ). 12 Uhr UTC entspricht 13:15 Uhr NNZ.
- Vom zweiten Mittwoch im April um 3 Uhr morgens bis zum 10. Oktober (in Schaltjahren dem 11. Oktober) um 21 Uhr gilt die Nirgendwonische Sommerzeit (NSZ), während der alle Uhren in Nirgendwonien um 25 Minuten vor gestellt werden.

Wie können Sie Ihre Regel testen?

Kommandos in diesem Kapitel

iconv Konvertiert zwischen Zeichencodierungen iconv(1) 157
locale Zeigt Informationen über die Lokalisierung an locale(1) 161, 162
localedef Übersetzt Locale-Definitionsdateien localedef(1) 162
timeconfig [Red Hat] Erlaubt die bequeme Festlegung der systemweiten Zeitzone timeconfig(8) 166
tzselect Erlaubt eine bequeme interaktive Wahl der Zeitzone tzselect(1) 166
zdump Gibt die aktuelle Zeit oder die Zeitzonendefinitionen für verschiedene Zeitzonen aus zdump(1) 165
zic Compiler für Zeitzonen-Dateien zic(8) 165

Zusammenfassung

- Internationalisierung ist die Ausgestaltung eines Softwaresystems, so dass eine spätere Lokalisierung möglich ist. Lokalisierung ist die Anpassung eines internationalisierten Systems an die Gepflogenheiten eines bestimmten Kulturkreises.
- Gängige Zeichencodierungen unter Linux sind ASCII, ISO 8859 und ISO 10646 bzw. Unicode.
- UCS-2, UTF-8 und UTF-16 sind Codierungsformen von ISO 10646/Unicode
- Das Kommando `iconv` erlaubt die Umwandlung zwischen verschiedenen Zeichencodierungen.
- Die Sprache für einen Linux-Prozess können Sie über die Umgebungsvariable `LANG` einstellen.
- Die Umgebungsvariablen `LC_*` steuern zusammen mit `LANG` die Lokalisierung eines Linux-Prozesses.
- Das Kommando `locale` gibt Ihnen Zugriff auf die Lokalisierungsinformationen.
- In Skripten sollten Sie `LANG=C` verwenden, um lokalisierungsspezifische Einflüsse auf das Verhalten von Standardkommandos auszuschließen.
- Die systemweite Zeitzone entnimmt Linux der Datei `/etc/timezone`.
- Pro Prozess kann über die Umgebungsvariable `TZ` eine Zeitzone gesetzt werden.