Linux II

Linux II

Was ist Encoding?

- Jedem benötigten Zeichen wird eine Zahl zugeordnet
- Welches Zeichen welcher Zahl zugeordnet wird, bezeichnet man als *Zeichencode*
- Das Format, in dem der Zeichencode (im Computer) repräsentiert wird, heißt *Encoding*

Zahlendarstellung im Computer

- Computer arbeiten binär (= zur Basis 2)
- Speicher: | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
- Ein Bit: | 1 | oder | 0 |
- Acht Bit werden ein Byte genannt und können 256 verschiedene Zustände repräsentieren (z.B. Zahlen zwischen 0 und 255)

1	1	1	0	0	0	1	1

1 Byte

Einschränkungen

- Wieviel Platz ein Zeichen bekommt, entscheidet darüber, wieviele Zeichen man unterscheiden kann
- Ist jedes Zeichen z.B. 4 Bit lang, können 16 Zeichen unterschieden werden
- Alle Zeichen sollten gleich viel Platz belegen; ggf. wird mit Nullen aufgefüllt

Linux II

ASCII

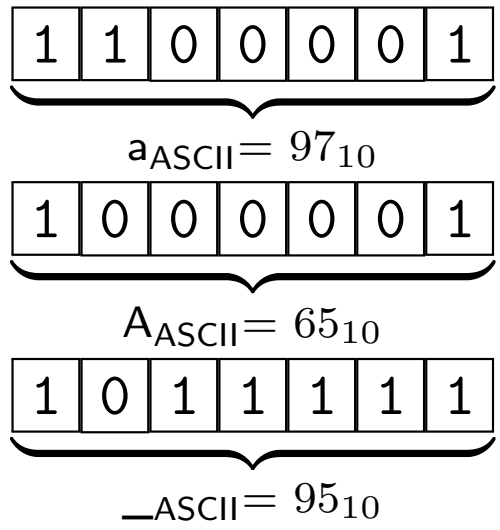
American Standard Code for Information Interchange

- Erstmals publiziert 1963
- Definiert $2^7 = 128$ Codepunkte
- ASCII enthält 33 nicht-druckbare und 95 druckbare Zeichen
- Alle Codepunkte können in 1 Byte gespeichert werden.
- Das 8. Bit dient als Kontrollsignal

Druckbare ASCII-Zeichen

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

Beispiele



Latin-1 / ISO 8859-1

ISO: International Organization for Standardization

- ASCII reicht nicht aus, um z.B. deutsche Umlaute darzustellen
- Erweiterung von ASCII: Latin-1
- In Latin-1 werden alle 8 Bit für 256 Zeichen genutzt
- Die meisten westeuropäischen Sprachen sind darstellbar
- Erste 7 Bit entsprechen denen in ASCII, achtes Bit sorgt dann für 128 zusätzliche Umlaute und Sonderzeichen

Beispiele

Latin-1 / ISO 8859-1

ISO: International Organization for Standardization

- ASCII reicht nicht aus, um z.B. deutsche Umlaute darzustellen
- Erweiterung von ASCII: Latin-1
- In Latin-1 werden alle 8 Bit für 256 Zeichen genutzt
- Die meisten westeuropäischen Sprachen sind darstellbar
- Erste 7 Bit entsprechen denen in ASCII, achtes Bit sorgt dann für 128 zusätzliche Umlaute und Sonderzeichen

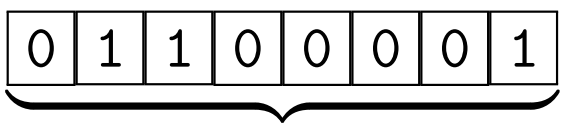
Beispiele

Latin-1 / ISO 8859-1

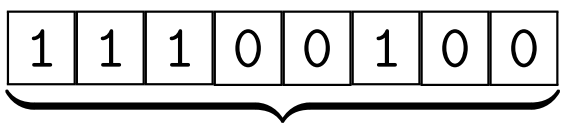
ISO: International Organization for Standardization

- ASCII reicht nicht aus, um z.B. deutsche Umlaute darzustellen
- Erweiterung von ASCII: Latin-1
- In Latin-1 werden alle 8 Bit für 256 Zeichen genutzt
- Die meisten westeuropäischen Sprachen sind darstellbar
- Erste 7 Bit entsprechen denen in ASCII, achtes Bit sorgt dann für 128 zusätzliche Umlaute und Sonderzeichen

Beispiele



$a_{\text{Latin-1}} = a_{\text{ASCII}} = 97_{10}$



$ä_{\text{Latin-1}} = 228_{10}$

Unicode

- Auch Latin-1 reicht nicht aus, um alle Zeichen darzustellen
- Im Chinesischen gibt es mehr als 47.000 verschiedene Zeichen
- 8-Bit-Zeichensätze bieten nur Raum für 256 Zeichen

Unicode

- Kodiert alle Zeichen dieser Welt
- Insgesamt 1.114.112 Codepunkte
- Verschiedene Encodings: UTF-8, UTF-16, UTF-32
- Unterschied zu Latin1: Anzahl der verwendeten Bytes kann variieren

Unicode

- Auch Latin-1 reicht nicht aus, um alle Zeichen darzustellen
- Im Chinesischen gibt es mehr als 47.000 verschiedene Zeichen
- 8-Bit-Zeichensätze bieten nur Raum für 256 Zeichen

Unicode

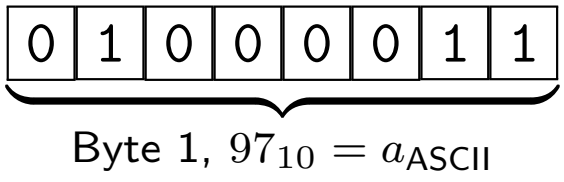
- Kodiert alle Zeichen dieser Welt
- Insgesamt 1.114.112 Codepunkte
- Verschiedene Encodings: UTF-8, UTF-16, UTF-32
- Unterschied zu Latin1: Anzahl der verwendeten Bytes kann variieren

UTF-8

- ASCII-kompatibel (ASCII-Zeichen werden so wie in ASCII kodiert)
- Nicht-ASCII-Zeichen sind 2-4 Byte lang
- Höchste Bits des ersten Bytes zeigen an, wie lang das Zeichen ist
  - $\boxed{0}$ $\rightarrow$ 1 Byte
  - $\boxed{1}\boxed{1}\boxed{0}$ $\rightarrow$ 2 Byte
  - $\boxed{1}\boxed{1}\boxed{1}\boxed{0}$ $\rightarrow$ 3 Byte
  - $\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{0}$ $\rightarrow$ 4 Byte
- Darauffolgende Bytes haben alle MSB auf 1 und das nachfolgende auf 0x (MSB: most significant bit, LSB: least significant bit)

UTF-8 – Beispiele

Beispiele

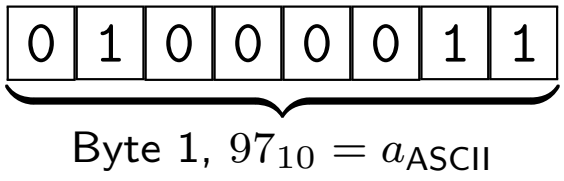
- 

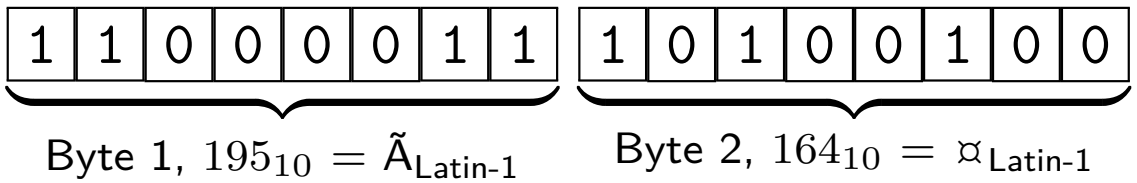
-

-

UTF-8 – Beispiele

Beispiele

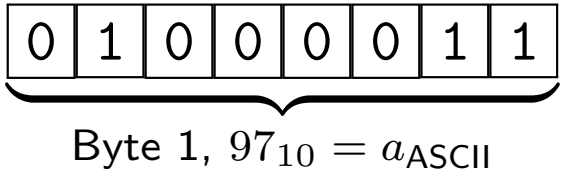
- 

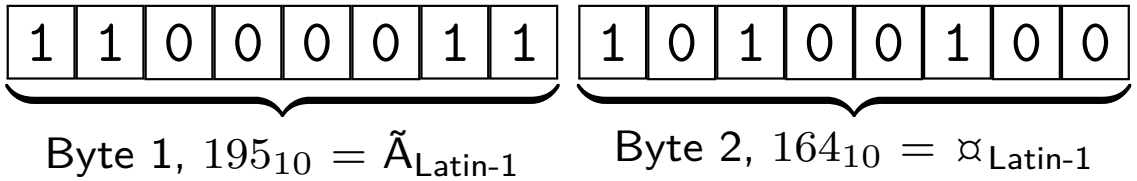
$= \text{a}_{\text{UTF-8}}$
- 

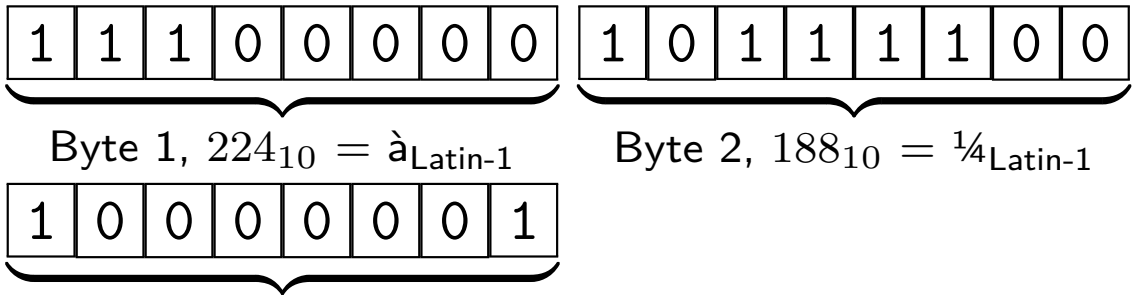
$= \text{ä}_{\text{UTF-8}}$
-

UTF-8 – Beispiele

Beispiele

- 

$= \text{a}_{\text{UTF-8}}$
- 

$= \text{ä}_{\text{UTF-8}}$
- 

Encoding erkennen

- Encoding erkennen ist schwer!
- Validierung von Bitfolgen als gültige oder ungültige Zeichen ist erst mit Unicode wieder möglich
- Viele Encodings haben Varianten, die in einzelnen Zeichen abweichen

Hex	Dez	8859-1	8859-2	8859-8	MacRoman
BE_{16}	190	¾	ž	¾	æ
CF_{16}	207	Ï	Ď		œ
$E4_{16}$	228	ä	ä		‰

Tabelle: Unterschiede zwischen Latin-1, Latin-2, Hebrew und MacRoman

Encoding erkennen

- Encoding erkennen ist schwer!
- Validierung von Bitfolgen als gültige oder ungültige Zeichen ist erst mit Unicode wieder möglich
- Viele Encodings haben Varianten, die in einzelnen Zeichen abweichen

Hex	Dez	8859-1	8859-2	8859-8	MacRoman
BE_{16}	190	¾	ž	¾	æ
CF_{16}	207	Ï	Ď		œ
$E4_{16}$	228	ä	ä		‰

Tabelle: Unterschiede zwischen Latin-1, Latin-2, Hebrew und MacRoman

Konvertierung

- Tools zur automatischen Konvertierung: **`recode`** , **`iconv`**
- Lesen *nicht* von der Standardeingabe
- Ein- und Ausgabe direkt über Dateien

Umwandlung aller Dateien im Verzeichnis (Latin-1 → UTF-8)

```
for file in $( ls directory )
    do
    recode l1..u8 $file
done
```

Linux II

Spracheinstellungen

Nicht nur Encoding – Nicht nur Sprache

- Sprache von Fehlermeldungen, Hilfetexte
- Default-Papiergröße für's drucken (DIN A4 oder US-Letter)
- Währungszeichen (€, $, ¥...)
- Dezimal- und Tausendertrennzeichen (1,000.00 vs. 1.000,00)
- Formatierung von Telefonnummern, Adressen, ...

Locale

- `locale` zeigt die aktuellen Einstellungen an

Spracheinstellungen

Nicht nur Encoding – Nicht nur Sprache

- Sprache von Fehlermeldungen, Hilfetexte
- Default-Papiergröße für's drucken (DIN A4 oder US-Letter)
- Währungszeichen (€, $, ¥…)
- Dezimal- und Tausendertrennzeichen (1,000.00 vs. 1.000,00)
- Formatierung von Telefonnummern, Adressen, …

Locale

- `locale` zeigt die aktuellen Einstellungen an

Spracheinstellungen in Umgebungsvariablen

- Spracheinstellungen werden in Umgebungsvariablen gespeichert

`LC_ALL`	Alle Einstellungen
`LC_CTYPE`	Zeichenklassifikation, Groß- und Kleinschreibung
`LC_COLLATE`	Sortierung
`LC_TIME`	Zeit und Datumsformat
`LC_NUMERIC`	Zahlenangaben
`LC_MONETARY`	Zahlenangaben mit Währung
`LC_MESSAGES`	Fragen, Fehlermeldungen, Dialoge
`LC_PAPER`	Papiergröße
`LC_NAME`	Namenformatierung
`LC_ADDRESS`	Adress- und Ortsangaben
`LC_TELEPHONE`	Telefonnummern
`LC_MEASUREMENT`	Maßeinheiten (Metric, Imperial, ...)

Zeichensatz und Schriftart

- Es reicht nicht, die Umgebungsvariablen zu ändern
- Die Schriftart (des Terminals) muss den Zeichensatz unterstützen
- Ändern der Schriftart:
  `Terminal` $\rightarrow$ `Set Character Encoding` / `Zeichenkodierung`

Ändern der Umgebungsvariablen

- ```
  :~$ export LC_LANG=de_DE.utf8
  ```
- ...
  ```

Linux II

AWK

- AWK ist eine Skriptsprache für strukturierte Textdaten
- GNU-Variante: GAWK
- AWK operiert auf zeilenorientierten Daten
- Jede Zeile repräsentiert ein Dokument, und jedes Dokument besteht aus mehreren Feldern.
- Die Felder sind standardmäßig durch Leerzeichen getrennt, andere Trenner sind möglich.
- Grundidee: Jede Zeile wird als ein Array aufgefasst:
  - `$0`: die komplette Zeile
  - `$1`: das erste Feld
  - ...

Grundlagen

- Ein AWK-Skript besteht aus Bedingungen und Anweisungen

```
Bedingung { Anweisungen }
```

- Wenn eine Bedingung erfüllt ist, wird der Anweisungsblock ausgeführt.
- Ist die Bedingung leer, wird die Anweisung immer ausgeführt.
- Ist die Anweisung leer, wird standardmäßig die gesamte Zeile ausgegeben, wenn die Bedingung greift.

Anweisungen

- Werden immer in `{ }` geschrieben
- Wenn keine Anweisung angegeben wird: `{ print $0 }`

Ausdruck	Funktionsaufruf, Wertzuweisung einer Variable (`{ VAR=0 }`)
Kontrollstruktur	**`if`**, **`for`**, ...
Ein- und Ausgabe-Statements	**`print`**, **`getline`**, ...(`{ print "/"$1 }`)
Deletion-Statements	Elemente aus Arrays löschen

`http://www.gnu.org/software/gawk/manual/html_node/Action-Overview.html`

Bedingungen

- Grundsätzlich: jeder `awk`-Ausdruck
- Wenn die Bedingung nicht den Wert 0 bzw. `""` hat, wird die Anweisung ausgeführt.
- Beispiel 1: Regulärer Ausdruck `$1 ~ /[a-c]/`
- Beispiel 2: Vergleich `$1=="word"`
- `BEGIN`: Die folgende Anweisung wird ausgeführt, *bevor* Daten gelesen werden.
- `END`: Die folgende Anweisung wird ausgeführt, *nachdem* alle Daten gelesen wurden.

`http://www.gnu.org/software/gawk/manual/html_node/Pattern-Overview.html`

Beispieldaten

```
290 india nnp
293 large jj
300 person nn
302 four cd
302 perform vbn
302 very rb
303 buddhist nnp
303 system nn
303 term nns
305 culture nn
```

Skriptaufruf

Auf der Kommandozeile

AWK-Skript muss in ' stehen:

```
:~$ gawk '$0 ~ /foo/ {print $1}' data.csv
```

In einer Datei

AWK-Skript kann aus einer Datei gelesen werden

```
:~$ gawk -f script.awk data.csv
```

Beispielskripte

material/sample1.awk

```
$3 == "nn" { print $2 }
```

Beispielskripte

material/sample1.awk

```
$3 == "nn" { print $2 }
```

material/sample2.awk

```
BEGIN { NOUNS=0 }
$3 == "nn" { NOUNS += $1 }
END { print NOUNS }
```

```
:~$ gawk -f material/sample2.awk material/data.csv
1827
```

Built-in Variables

AWK verwendet einige eingebaute Variablen, hier nur zwei wichtige Beispiele:

- FS *field separator* – Zeichen oder regulärer Ausdruck
- NR *number of records* – wieviele Zeilen bereits gelesen wurden

Field Separator ändern

```
:~$ gawk 'BEGIN{FS="\t"}{print $1}'
```

www.gnu.org/software/gawk/manual/gawk.html#Built_
002din-Variables

Weitere Beispiele

Nützliche AWK-Einzeiler

- `http://www.pement.org/awk/awk1line.txt`
- `http://www.catonmat.net/blog/awk-one-liners-explained-part-one/`

Übung 3