

## Techniken der CL in Perl

Reitter, 2002

Reitter: Perl, 2002

## Termine

- 26.4. 11.00 (s.t.) – 16.15
- 17.5. 11.00 (s.t.) – 16.15
- 14.6. 11.00 (s.t.) – 17.15
- 12.7. 11.00 (s.t.) – 16.15

Reitter: Perl, 2002

## Perl für Linguisten

- Programmieren in Perl
- Arbeit mit Korpora
- Statistische Grundlagen
- Einführungs- und Programmierkurs
- Voraussetzung: Unix-Account

Reitter: Perl, 2002

## Kurs-Kontakt

[www.davids-welt.de/perl](http://www.davids-welt.de/perl)

E-Mail:

[reitter@ling.uni-potsdam.de](mailto:reitter@ling.uni-potsdam.de)

Reitter: Perl, 2002

## Scheine..

- Referat (45 - 60 Minuten) mit Ausarbeitung (5 – 10 Seiten)
- oder Projekt mit Vorstellung (30 - 60 Minuten)

Reitter: Perl, 2002

## Referate

- Reguläre Ausdrücke (17.5.)
- Arrays, Hashes, Stacks (26.4.)
- Datei-E/A, Pipes (17.5.)
- XML in Perl (14.6.)
- Das Common Gateway Interface; Zusammenspiel mit Webserver und das Modul CGI::Simple (14.6.)
- Objektorientierung in Perl (optional) (14.6.)
- Statistische Grundlagen und Korpuslinguistik (2 Personen) (14.6. / 12.7.)
- Part-of-Speech Tagging in Perl (Projekt) (12.7.)
- Suchmaschine in Perl (Projekt, 2 Personen) (12.7.)

Reitter: Perl, 2002

## Literatur

- [www.perlmonks.org](http://www.perlmonks.org)
- [www.perl.com](http://www.perl.com)  
(Standard-Doku:  
<http://www.perldoc.com/perl5.6/pod/perl.html>)
- [www.cpan.org](http://www.cpan.org)  
(Module zum Download)
- Schwartz / Phoenix: Learning Perl. O'Reilly 2001
- Wall/Christiansen/Orwant: Programming Perl. O'Reilly 2000
- Folien / Material in Anlehnung an Feddes 1999

Reitter: Perl, 2002

## Arrays, Hashes, Stacks

- Datenstrukturen (informell)
- @Arrays, @Lists: [], grep, sort
- @Stacks: push, pop, shift, unshift, split
- %Hashes: {}, delete, each, exists, keys, values
- Beispiel-Implementation(en)

Reitter: Perl, 2002

## Datei-E/A, Pipes

- Ein- und Ausgabe (open, close, print), Filehandles, Verzeichnisoperationen, Dateiinformationen (exists, -d)
- Einseitige Prozess-Pipes (open, print)
- stderr-Ausgaben (warn)

Reitter: Perl, 2002

## CGI

- Architektur (Client – Server – Servlet)
- CGI: Schnittstellen-Dokumentation
- Umgebungsvariablen
- CGI::Simple - Modul
- Beispielimplementation eines Perl-CGIs
- Überblick: Performance-Probleme, typische Installationsprobleme, CGI::Fast

Reitter: Perl, 2002

## XML-Parsing

- Kurzeinführung in XML (XML, DTD – kein Schema)
- Relationen XML, SGML, HTML, VoiceXML
- Ein XML-Parsing-Modul (z.B. Expat)
- Beispielimplementation

Reitter: Perl, 2002

## Statistische Grundlagen und Korpuslinguistik

- Wahrscheinlichkeiten, Bayes-Formeln
- Klassifizierungs-Prinzipien (Merkmale, Klassen)
- Prinzipien Korpus-Linguistik und Maschinelles Lernen („Lernen“, Evaluierung)
- Signifikanz und Signifikanz-Tests (Student's t-test, chi-square test)
- Übersicht über Standard-Korpora (Penn Treebank, Celex, MULTEXT, etcetc)
- Jeweils kurze Beispielimplementation
- Literatur: Manning&Schütze: *Statistical Foundations for Natural Language Processing*. MIT Press

Reitter: Perl, 2002

## Suchmaschine

- Indizieren (Dateiverzeichnisse, HTML-Parsing, Datenstrukturen, Speichern)
- Suchen (CGI, HTML)
- 2 Personen, Vorstellung: letzter Termin

Reitter: Perl, 2002

## Perl als Programmiersprache

- „Practical Extraction and Report Language“, Larry Wall
- Man muss nicht viel verstehen, um anfangen zu können (BASIC-Effekt vs. C++)
- Schnelle Prototypenimplementierung
- Praktische Modul-Bibliotheken
- kostenlos und open-source
- Gute Zeichenketten-Bearbeitung mit regulären Ausdrücken -> gutes Werkzeug für Linguisten

Reitter: Perl, 2002

## Perl vs. Prolog

- Imperative Programmierung; Variablenzuweisung temporär; Kontrollstrukturen
- Kaum interaktive Programme
- Stringverarbeitung
- viele Bibliotheken erhältlich

Reitter: Perl, 2002

## Perl vs. C++

- Schnelle Einarbeitungszeit
- Just-In-Time-Compilierung
- kaum Typisierung
- Reguläre Ausdrücke
- Imperativ, Objektorientierung möglich
- Modularität nur begrenzt möglich

Reitter: Perl, 2002

## Perl zu Hause

- Linux: Fast immer schon installiert. Editor: zB *xEmacs*
- MacOS: Bei OS X schon dabei; sonst: <http://www.macperl.org/>
- Windows: ActivePerl (kostenlos) <http://www.activestate.com/Products/ActivePerl/> IDE: *OpenPerlIDE* oder *Komodo*

Reitter: Perl, 2002

## Perl nutzen

- Einloggen in Unix mit `telnet` oder `Exceed`
- `hyperion` oder `helios`
- Ausführbare Dateien unter Unix:  
`chmod +x dateiname.pl`  
und erste Zeile: `#!/usr/bin/perl`  
Aufruf: `./dateiname.pl`
- Oder Aufruf mit:  
`perl -w dateiname.pl`

Reitter: Perl, 2002

## Aufgabe

```
# Beispiel
print "Hello World!\n";
$day = 'Freitag';
print "Heute ist $day.\n";
$daily = substr($day, 1,
                4)."täglich";
print "$daily findet dieser Kurs
      statt.";
```

Reitter: Perl, 2002

## Grundlagen: Strings

- `$a = "Knuth";`
- `$b = "Guten Tag, Herr $a!";`
- `$c = $b."\n Wie geht's?";`
- `Print 'Zeilenvorschub mit \n';`

Reitter: Perl, 2002

## Grundlagen

- Skalare (Variablen): `$test`
- Befehle: `print`
- Gearbete Strings: `„\n“`
- Kontrollstrukturen: `if, elsif, else`
- Operatoren: `eq, ne, >, <, !, ==`
- Kontrollstrukturen: `for, while`
- Funktionen mit `sub`
- Der Kontext (`$_`)
- Lokale Variablen: `my`

Reitter: Perl, 2002

## Grundlagen: Operatoren

- `+ - * /`
- `%` (Modulo)
- String-Konkatenation: `.`  
z.B. `"ein"."und"."vierzig"`

Reitter: Perl, 2002

## Listen

- `@liste = (2,3,5,7,11,13);`
- `print "erstes Element: ", $liste[0];`
- `print "alle Elemente: ", @liste;`
- `Push @liste, 17;`  
`print shift(@liste);`
- `$liste[7] = 19;`

Reitter: Perl, 2002

## Hashes

```
%hash = (david => '79708990',
         andrea => '01743173368',
         sven => '42020577');

$hash{handy} = '01714160693';
print "Andrea: ", $hash{andrea};
• foreach(values %hash) {print $_;}
```

Reitter: Perl, 2002

## Variablen im Kontext

- Variablen sind \$skalare und @listen und %hashes
- Automatische Umwandlung ist möglich:
  - Liste -> Skalar (Länge der Liste)
  - Hash -> Liste (jeweils Elementpaare)
  - Hash -> Skalar (Anzahl genutzer / reservierter Speicherplätze = *Buckets*)

Reitter: Perl, 2002

## Variablen im Kontext #2

- Beispiele für Kontext: siehe `kontext.pl`

Reitter: Perl, 2002

## Referenzen

- Referenzen sind Zeiger auf Variablen, d.h. auf Skalare, Listen oder Hashes.
- Referenzen erhält man durch Referenzierung:

```
$ref = \@meineListe;  
$ref2 = \%meinHash;
```

Reitter: Perl, 2002

## Dereferenzierung

- Interpretation einer Referenz als Array-Referenz:

```
@array = @{$ref};  
$element = ${$ref}[4];
```
- Als Hash-Referenz:

```
%hash = %{$ref};  
$value = $hash{andrea};
```

Reitter: Perl, 2002

## Beliebte Fehler #1

Ein Array von Arrays?

```
foreach $i (1..10) {  
    @array = messwerte($i);  
    $AofA[$i] = @array;  
}
```

Reitter: Perl, 2002

## Beliebte Fehler #2

Ein Array von Integern!

So ist's deutlicher:

```
foreach $i (1..10) {  
    @array = messwerte($i);  
    $counts[$i] = scalar @array;  
}
```

Reitter: Perl, 2002

## Beliebte Fehler #3

Ein Array von Array-Referenzen?

```
my @AoAR;
foreach $i (1..10) {
    @array = messwerte($i);
    $AoAR[$i] = \@array;
}
```

Reitter: Perl, 2002

## Beliebte Fehler #4

So geht 's

```
foreach $i (1..10) {
    @array = messwert($i);
    $AoA[$i] = [ @array ];
}
```

**Merke: [ ] erzeugt eine Kopie des Arrays und gibt eine Referenz darauf zurück!**

Reitter: Perl, 2002

## Aufgabe Datentypen

Schreiben Sie ein Perl-Programm, das Kardinalzahlen (0-99) in Zahlenausdrücke (in einer europäischen Sprache) wandelt. Zum Beispiel: 35 – „fuenfundreissig“.

Benutzen Sie: Skalare (Strings), Hashes, String-Konkatenation mit '.', Funktionen mit 'sub', if-Abfragen.

Reitter: Perl, 2002

## Der Kontext

- (Nicht verwechseln mit dem "Kontext", in dem eine Variable als Skalar oder Liste verwendet wird!)
- `$_` ist der Default-Skalar, `@_` (mit Zugriff als `$_[0]`, `$_[1]`...) die Default-Liste. zB sind gleichbedeutend:  
`$parameter = shift;`  
`$parameter = shift @_;`

Reitter: Perl, 2002

## Kontext #2

```
foreach (1..10)
{
    print $_, "\n";
}
```

Reitter: Perl, 2002

## Aufgabe RegExp

- Alle Wörter in einem Text, die mit Großbuchstaben beginnen
- Alle Sätze eines Textes
- Ersetze mehrfache Leerzeichen durch einzelne
- Finde alle URLs in einer Datei und drucke diese aus

Reitter: Perl, 2002

## Datei Ein- und Ausgabe

Reitter: Perl, 2002

## Ein/Ausgabe

- Unix-Modell: Ein Programm liest Daten von der einen Seite und schreibt Daten auf die andere.
- Die Seiten haben Namen:  
Leseseite: "stdin"  
Schreibseite: "stdout"

Reitter: Perl, 2002

## Ausgabe

- Ausgeben auf die stdout-Seite:

```
print "Hallo ! \n";  
print $a, $b;
```

Reitter: Perl, 2002

## Eingabe

- Liest eine Zeile:

```
$eingabe = <STDIN>
```

- Liest alle Zeilen:

```
@eingabe = <STDIN>
```

Reitter: Perl, 2002

## Piping

- In Unix kann man die Ausgabe des einen Programms als Eingabe für ein anderes verwenden:

```
ls | grep txt  
cat MANUAL | grep beenden
```

Reitter: Perl, 2002

## Piping

- Wenn unser Programm mit

```
@eingabe = <STDIN>
```

bis zum "Ende" liest, dann geht das nur per Piping:

```
cat README | ea.pl
```

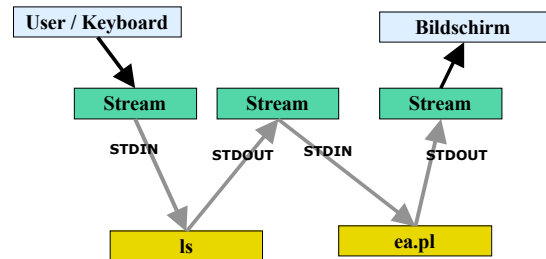
Reitter: Perl, 2002

## Streams

- STDIN, STDOUT bezeichnet man als "Streams"
- Stream: Abstraktes Objekt, das Clients (Prozessen) das Lesen und Schreiben von Text (o.a.) ermöglicht
- STDIN: Betriebssystem schreibt (Tastendrucke zB), unser Prozess liest
- STDOUT: Unser Prozess schreibt, Betriebssystem liest (und gibt zB auf Bildschirm aus)

Reitter: Perl, 2002

## Streams #2



Reitter: Perl, 2002

## Zusätzliche Streams öffnen

- STDIN und STDOUT sind immer offen.
- Zusätzlich können auch Streams mit anderen Quellen geöffnet werden: Dateien.
- Öffnen: `open STREAM, FILECODE;`
- Schließen: `close STREAM;`

Reitter: Perl, 2002

## Open / Close - Syntax #2

- Zum Schreiben öffnen:

```
open OUTPUT, ">/home/dr/datei.txt";
print OUTPUT "Hello World.";
close OUTPUT;
```

Reitter: Perl, 2002

## Open / Close - Syntax

- Zum Lesen öffnen:

```
open INPUT, "</home/dr/datei.txt";
@dateinhalt = <INPUT>;
close INPUT;
```

Reitter: Perl, 2002

## Zeilenweise lesen

- <INPUT> im Listen- wie im Skalar-Kontext enthält die Zeile(n) inklusive des Zeilenende-Zeichens!

```
while(<INPUT>)
{
    my $zeile = $_;
    $zeile =~ /\n/;
    push @dateinhalt, $zeile;
}
```

Reitter: Perl, 2002

## Ausgabe

- Diese Befehle bewirken das gleiche:

```
print STDOUT "Hallo";  
print "Hallo";
```

Reitter: Perl, 2002

## Übung: E/A

- Schreiben Sie ein Programm, das alle Zeilen einer Datei liest und nummeriert wieder ausgibt.

Der Name der Datei soll in `$ARGV[0]` stehen. Das ist das erste Argument. Ihr Programm kann wie folgt aufgerufen werden:

```
ea.pl datei.txt
```

Reitter: Perl, 2002

## Übung: E/A 2

- Schreiben Sie Ihr Programm so um, dass es von STDIN liest, sofern das erste Argument '-' (Minus-Zeichen) ist.

Reitter: Perl, 2002

## STDERR

- STDERR ist ein weiterer Standard-Stream, der Fehlermeldungen dient.

```
print STDERR "Programmfehler";  
warn "Programmfehler!";
```

Reitter: Perl, 2002

## Zum Anhängen öffnen

```
open OUTPUT, ">>/home/dr/logfile";
```

Reitter: Perl, 2002

## Prozesskommunikation

- Streams haben einen sogenannten "Cache". Erst wenn mehrere Zeilen ausgegeben wurden, werden sie tatsächlich weitergeleitet.
- Der Befehl `flush HANDLE;` sorgt dafür, dass der übergebene Code sofort weitergeleitet wird.

Reitter: Perl, 2002

## Übung: E/A #3 - grab

- Schreiben Sie ein Programm, das einen Text von STDIN liest und zeilenweise auf ein Vorkommen eines Wortes untersucht. Dieser wird in \$ARGV[0] übergeben. Zeilen, die das Wort enthalten, sollen nach STDOUT ausgegeben werden. Aufruf so:  
  
cat datei.txt | grab.pl hallo
- Das Programm soll auch Vorkommen der Pluralform des Wortes (englischer Plural) finden!

Reitter: Perl, 2002

## Verzeichnisoperationen

- Existenz einer Datei prüfen
- Ist eine Datei schreibgeschützt?
- Verzeichnis einlesen
- Welche Einträge sind Ordner?

Reitter: Perl, 2002

## Dateien

- **-e** DATEINAME  
wahr, gdw. Datei existiert
- *Jeweils für den aktuellen Nutzer:*
- **-r** DATEINAME  
wahr, gdw. Datei lesbar
- **-w** DATEINAME  
wahr, gdw. Datei beschreibbar ist
- **-x** DATEINAME  
wahr, gdw. Datei ausführbar ist

Reitter: Perl, 2002

## Verzeichnisse

- Alle passenden Dateien und Ordner für einen Wildcard in einem Verzeichnis sammeln:

```
@dateiliste = glob("*.txt")
```

- Test, ob ein Eintrag ein Ordner ist:

```
print "Ordner: " if (-d $datei);
```

Reitter: Perl, 2002

## Pfade

- open INPUT,  
"</home/maria/lexikon";

(auch unter Windows sind /Slashes statt \Backslashes zu verwenden!)

Reitter: Perl, 2002

## Dateien löschen

- Datei löschen:

```
unlink $datei1, $datei2, ... ;  
unlink @dateien;
```

- Verzeichnis löschen:

```
rmdir $verzeichnis;
```

Reitter: Perl, 2002

## Module

- Perl-Bibliotheken nennt man *Module*
- Umfassendes Modulverzeichnis:  
[www.cpan.org](http://www.cpan.org)
- Module verwenden:
  - `use Getopt::Std;`

Reitter: Perl, 2002

## Getopt-Modul

```
use Getopt::Std;
my %opts = ();
getopts('i:o:I:O:', \%opts);
my $infile = $opts{'I'} || 'infile';
my $outfile = $opts{'O'} || 'outfile';
my $inform = $opts{'i'} || 'swiss';
my $outform = $opts{'o'} || 'fasta'
```

Reitter: Perl, 2002

## Getopt #2 - Einfacher

```
use Getopt::Std;
getopts('f:plgh');
if ($opt_f) {
    $format = $opt_f;
} else {
    $format = "Genbank";
}
```

Reitter: Perl, 2002

## Module installieren

- Modul Download: [www.cpan.org](http://www.cpan.org)
- Meistens:

```
perl Makefile.pl
make
make install
```

Reitter: Perl, 2002

## Lokale Module

- Wenn kein root-Zugriff, dann Angabe eines Library-Pfades - siehe Informationsdatei wie z.B. `INSTALL`, `README`
- In Programmen Angabe des lokalen Pfades:

```
use lib "/home/dr/perl";
use lib "~/perl";
```

Reitter: Perl, 2002

## XML/HTML in Perl

- Einfache Aufgaben mit Regulären Ausdrücken zu lösen:

```
if ($doc =~ /<body>(.*?)</body>/is)
{ $doc_body = $1; }
```

Reitter: Perl, 2002

## XML Parsing

- Für komplexe Aufgaben brauchen wir einen Parser:

```
use XML::Parser;

$p1 = new XML::Parser();
$p1->parsefile('maz-12345-
korruption.xml');
```

Reitter: Perl, 2002

## XML::Expat

- Handler definieren:

```
$p2 = new XML::Parser(Handlers =>
{Start => \&handle_start,
End => \&handle_end,
Char => \&handle_char} );

sub handle_start {...};
sub handle_end {...};
sub handle_char {...};
```

Reitter: Perl, 2002

## XML::Expat

```
sub handle_start {
    my ($expat, $element, %attrs) = @_;

    $line_number = $expat->current_line;

    print $element, ": ";
    foreach(keys %attrs) {
        print $_."="."$attrs{$_}." ";
    }
}
```

Reitter: Perl, 2002

Source: [www.webreference.com](http://www.webreference.com)

## Parsing mit Expat

- Übung: Lesen Sie die Datei maz.xml und legen Sie alle enthaltenen <segment id = "XY"> Tags in einem Hash ab:  
  
\$hash{\$id} = Segment-Text
- Merken Sie sich die jeweilige ID (im Start-Handler) und löschen Sie sie wieder im End-Handler. Schreiben Sie den Text im Char-Handler.

Reitter: Perl, 2002