Ruby in seiner Laufzeitumgebung
Scripting-Komfort

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Scripting-Komfort

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Scripting-Komfort

Hintergrund

- Ausgangspunkt: Unix-Tradition
  - Der Unix-“Werkzeugkasten“
  - Verbinden vieler kleiner „Werkzeuge“ mit Shell-Skripten / Pipes
  - Ergebnis: „Einzeiler“ zum Lösen vieler Alltagsaufgaben
  - Beispiel-Tools: grep, awk, sed, wc, ps, kill, …
- Vorbild Perl
  - Ersatz des traditionellen Unix-Ansatzes durch Perl-“Einzeiler“
  - Vorteil: Eine Sprache / ein Werkzeug für „alles“
  - Beispiel: Einmal Perl lernen statt Shell-Syntax + awk-Syntax + sed + … genügt
- Die Folge
  - Ruby orientierte sich an Perl und schuf analoge Möglichkeiten
  - Verallgemeinerter Anspruch: Ruby als „***glue language***“ mit vielen Möglichkeiten zur Anwendungs-Integration

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Scripting-Komfort

Ruby besitzt zahlreiche **Programmoptionen**. Einige erleichtern das Schreiben einzeiliger, wirkungsvoller Kommandos:

Option -a

Wirkt wie `$F = $_.split`

Wird in Verbindung mit -n bzw. -p eingesetzt. Siehe Option -F

Option -n

Wirkt wie `while gets; ...; end` um den angegebenen Code:

```
# Wirkt wie "grep":
$ ruby -n -e "print if /root/" /etc/passwd
# Wirkt wie "cut":
$ ruby -a -n -F':' -e 'puts $F[6]' /etc/passwd
```

Option -p

Wirkt wie `while gets; ...; print; end` um den Code in -e:

```
# Wirkt wie "tr":
$ echo $HOME| ruby -p -e '$_.tr!("ehlw","EHLW")'
/LocaL0/WErntgEs
```


© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Scripting-Komfort

Zugriff auf Umgebungsvariablen:

- Sehr einfach durch ENV
  (hinter dem sich ein *Singleton* der Klasse Object verbirgt).
- ENV verhält sich weitgehend wie ein Hash, ist aber keiner!
- Bei Bedarf in Hash umwandelbar mit Methode to_hash
- Beispiele:

```
$ echo $HOME                    # Shell-Ausgabe
/local0/werntges
$ ruby -e 'puts ENV["HOME"]' # Analog in Ruby
/local0/werntges                # Nun Ändern in Ruby:
$ ruby -e 'ENV["HOME"]="/var/tmp"; puts ENV["HOME"]'
/var/tmp
$ echo $HOME  # Wert im Elternprozess nicht geändert:
/local0/werntges
```

```
ENV["PATH"] += ":/my/bin"  # Bequemes Arbeiten in Ruby
```

(Beispiel in `fork`-Demo, später!)

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Scripting-Komfort

Zugriff auf Kommandozeilen-Optionen und -Parameter:

- Sehr einfach durch Array ARGV
- Alle von Ruby selbst nicht verbrauchen Werte finden sich in ARGV.
- Beispiele:

```
$ cat argv.rb
ARGV.each {|arg| puts arg}
$ argv.rb -x -f myfile
-x
-f
myfile
```

```
$ ruby -v argv.rb -v -x
ruby 1.8.5 (2006-08-25) [i686-linux]
-v
-x
```

- ACHTUNG - anders als in "C" ist ARGV[0] nicht das aktuelle Kommando (das befindet sich in $0), sondern das erste Argument!
- Hinweis: Bequeme Verarbeitung von Optionen mit **GetoptLong**

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Scripting-Komfort

Automatisches Lesen von Quelldaten:

- Zeilenweises Einlesen aus einer Datei, mehreren Dateien oder `stdin` ist eine häufige Tätigkeit. Ruby bietet dafür einen Automatismus an: ARGF
- **ARGF** bietet Zugriff auf die (virtuelle) Konkatenierung aller Dateien, die per Kommandozeilen-Argument übergeben wurden.
- Sind keine angegeben, wird ARGF mit `stdin` identifiziert.
- **ARGF** (Synonym: **$<**) ist ein spezielles Objekt mit Methoden wie in Klasse `File`. Auch vorhanden: **fileno, filename, lineno, pos**.
- Methode **Kernel.gets** macht davon Gebrauch. Sie können also Textdateien einlesen & verarbeiten, ohne sie explizit zu öffnen und zu schließen:

```
# Wirkt wie "wc -l *.rb":
$ ruby -e 'cw=0; while gets; cw+=1; end; puts cw' *.rb
```

```
$ cat *.rb | ruby -e 'cw=0; while gets; cw+=1; end; puts cw'
$ cat *.rb | wc -l     # Sollte denselben Wert liefern
```


© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Variablen

... teils aus der Perl-Tradition,
teils Ruby-eigene

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Variablen

- Darstellung
  - Die Darstellung erfolgt nach Sachgebieten
  - Jeder Eintrag beginnt mit einer (komma-separierten Liste synonymer) Variablen, gefolgt vom
    1) Klassennamen des Objekts und
    2) (optional) in Klammern einer Default-Belegung.
  - Nächste Zeile: Erläuterungen, stichwortartig
  - Beispiel:

```
$>, $defout                IO   ($stdout)
```

    Ziel von Kernel#print / printf

- Inhalt
  - Erläuterungen sind sehr knapp. Primäres Ziel ist zu zeigen, was existiert.

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Variablen

- ***Exception handling***

  $! `Exception`
  Das letzte Ausnahmeobjekt

  $@ `Array`
  Stack backtrace;
  vgl. Exception#backtrace

- **Separatoren**

  $/, $-0 `String (newline)`
  Zeilentrennzeichen, wie von gets, readline verwendet.

  $ `String (nil)`
  Output record separator, wie von print und IO#write verwendet.

  $, `String (nil)`
  von print und Array#join zwischen Parametern verwendet

  $;, $-F `String (nil)`
  Trennzeichen für split

- ***Input/Output***

  $stdin `IO (STDIN)`
  $stdout `IO (STDOUT)`
  $stderr `IO (STDERR)`
  Selbsterklärend

  $. `Fixnum`
  Nummer der zuletzt gelesenen Zeile der aktuellen Eingabedatei. Entspricht ARGF.lineno

  $< `Object`
  Synonym von ARGF, s. dort
  Ähnlich wie ein File-Objekt!

  $>, $defout `IO ($stdout)`
  Ziel von Kernel#print / printf

  $FILENAME `String`
  entspricht ARGF.filename

Demo
„print_like_python.rb“

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Variablen

- **Laufzeitumgebung**

  $0 String
  Name des gerade laufenden Ruby-Programms

  $$ Fixnum
  ... und seine PID

  $? Fixnum
  Exit-Status (u.a. errno) des zuletzt beendeten Prozesses

  $:, $LOAD_PATH Array
  Die Verzeichnisse, in denen load und require Dateien erwarten.

  $" Array
  Array mit Namen der von require geladenen Dateien.

  $SAFE Fixnum
  Sicherheitsstufe (0), 0-4

  __FILE__ String,
  __LINE__ String
  Name und aktuelle Zeilennummer der Quelltextdatei.

- **Kommandozeile**

  $DEBUG Object
  true, falls -r oder --debug

  $VERBOSE, $-v, $-w Object
  true, falls -v, -w oder --verbose gesetzt

  $F Array
  erhält Ergebnis von split, wenn -a und (-p oder -n) gesetzt.

  $-a / $-l / $-p Object
  true, wenn -a / -l / -p gesetzt

  $-x
  Wert der Option -x (0, a, d, F, i, K, l, p, v)

  $* Array
  Synonym von ARGV

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Variablen

- **Variablen, die von $~ abhängen und auf die nicht zugewiesen werden darf**:
  - `$1, $2, $3, ...`
    entspricht `$~[1], $~[2], ...`
  - `$&` Letzter Trefferbereich
  - `$\`` String vor dem letzten Trefferbereich
  - `$'` String hinter dem letzten Trefferbereich
  - `$+` Inhalt der letzten Regex-Gruppe des letzten "match"
- **Bemerkungen**
  - Alle o.g. Var sind String-Objekte.
  - Alle Var. werden von einem erfolglosen `=~` auf `nil` gesetzt.
  - Sonderfall, änderbar:

    `$=` `Object (nil)`

    Auf `true` setzen: Entspricht /.../i

- **Lokale Variablen**
  - `$_` Die zuletzt von `gets` oder `readline` eingelesene Zeile im aktuellen *scope*.
  - `$~` Das zuletzt erzeugte `MatchData`-Objekt

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Konstanten

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Globale Konstanten

- Elementare Konstanten
  - **TRUE** Synonym von true
  - **FALSE** Synonym von false
  - **NIL** Synonym von nil
- Spezielles
  - **DATA**
    Gestattet das Einlesen von Zeilen hinter __END__ im Quelltext, falls vorhanden.
  - **TOPLEVEL_BINDING** Binding
    (binding-Konzept ausgelassen)
  - **RUBY_PLATFORM** String
    z.B. "i686-linux"
  - **RUBY_RELEASE_DATE**
  - **RUBY_VERSION** String
    z.B. "1.8.7"

- Konst. für die Kommandozeile
  - **ARGF, $<** `Object`
    ARGF besitzt ähnliche Methoden wie ein File-Objekt!
    Lesen von ARGF bewirkt sequenzielles Lesen von allen Dateien, die als Kommando-zeilen-Argumente angegeben wurden, Lesen von $stdin sonst.
    Analogie zu Perl, „<>“ (diamond)
  - **ARGV, $*** `Array`
    Array der Argumente der Kommandozeile
  - **ENV** `Object`
    Zum hash-artigen Zugriff auf Umgebungsvariablen.
  - **STDIN, STDOUT, STDERR** `IO`
    Die festen Standard-Streams als Konstanten, Default-Werte von $stdin, $stdout, $stderr.

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Standardklassen

Ruby's eingebaute Klassen:
Eine Übersicht

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Die Standardklassen: Übersicht

- **Numerische Klassen**
  - Bignum, Fixnum, Integer, Float, Numeric
- **I/O-Klassen**
  - Dir, IO, File, File::Stat
- **"Basistypen"**
  - Array, Hash, Range, String, Symbol
- **Boole'sche Klassen etc.**
  - TrueClass, FalseClass
  - NilClass
- **Reguläre Ausdrücke**
  - Regexp, MatchData, MatchingData (Synonym)
- **OO-Klassen**
  - Binding, Class, Method, Module, Object, UnboundedMethod
- **Strukturbildung**
  - Struct, Struct::Tms
- **Code-Ausführung**
  - Continuation, Proc, Thread, ThreadGroup
- **Systemnahe Klassen**
  - Exception, Time

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Die Standardmodule: Übersicht

- **Comparable**
  - Vergleichsoperatoren
- **Enumerable**
  - Viele praktische Methoden auf Basis von „each“ und <=>
- **Errno**
  - kapselt Fehlercodes des Betriebssystems
- **FileTest**
  - Alternative Methoden zu denen aus File::Stat
- **GC**
  - Garbage Collector
- **Kernel**
  - Zahlreiche Methoden rund um "libc"
- **Marshal**
  - Objektpersistenz
- **Math**
- **ObjectSpace**
  - Für GC
- **Precision**
- **Process**
  - Unter-Einheiten: **GID,Sys,UID**
  - Verwaltung der OS-Prozesse
- **Signal**
  - Signalaustausch zw. Prozessen

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden

Neue Klassen, Module & Exceptions in Ruby 1.9

- **BasicObject**
  - Die neue Basisklasse
  - Die meisten Klassen werden weiterhin von „Object“ abgeleitet!
- **Fiber**
  - „Semi-coroutines“, verwandt mit Threads
  - Genutzt für die Umwandlung von internen in externe Iteratoren
- **FiberError**
  - Zugehörige Fehlerklasse
- **KeyError**
  - Ein spezieller IndexError
- **Mutex**
  - „*mutual exclusion*“ – zur Unterstützung von *Multithreading*
  - War bisher schon in StdLib
- **StopIteration**
  - Teil von „Enumerator“
  - Noch nicht dokumentiert
- **VM**
  - Noch nicht dokumentiert; Folge der VM YARV?

© 2003, 2008 H. Werntges, FB Design Informatik Medien (DCSM), FH Wiesbaden