Fortgeschrittene Perl-Techniken

Perl Community-Features in einfachen und komplexen Umgebungen nutzen

Über den Autor

* Alvar C.H. Freude
  * Freiberuflicher Trainer, Software-Entwickler, Berater
  * Diplom-Kommunikations-Designer (FH)
  * http://www.perl-blog.de/
  * http://alvar.a-blast.org/
  * alvar@a-blast.org

Intro

* Viele gute Werkzeuge aus der Perl-Community
  * In Unternehmen oft nicht oder nur teilweise genutzt
  * Erleichtern aber Arbeit
  * Machen vieles erst möglich
  * Sorgen für robusten, wartbaren und effizienten Code

Für jeden Umfang das Passende

- Programme in Perl können entsprechend für jeden Umfang geschrieben werden
  - Klein: der Einzeiler auf der Kommandozeile
  - Mittel: Perl-Skripte
  - Wiederverwendbar bzw. groß bis sehr groß: ganze Applikationen in Paketen und Modulen

Der Einzeiler

- Besonders geeignet für *„Write Only Code“*
  - Für kleine Hacks zwischendurch
  - Kleine Wegwerf-Aufgaben, Shell-Skripting

```
perl -i.bak -npe 's(Helene Graf)(Helene Graf-Drakula)g' *.txt

perl -e 's/.bak$// and unlink "$_.txt" for @ARGV' *.bak

perl -nE '$count++; END {say $count}' *.pm */*.pm
```

Perl-Skripte

* Perl-Programme theoretisch beliebiger Größe
* Geeignet besonders für kleinere Sachen, die Sie an einem Nachmittag schreiben
* Wenig geeignet zum Wiederverwenden
* Bitte keine Perl-Skripte mit mehreren tausend Zeilen Code schreiben:
  * Wird schnell schwer wartbar und unübersichtlich

Strict-Mode für sauberen Code

Hinweis:
Jedes Perl-Programm, das nicht nur ein kleiner dreizeiliger Hack ist, sollte **immer** *Warnungen* und den *Strict-Modus* anschalten. Das verhindert Probleme und hilft, Fehler schneller zu finden:

```
#!/usr/bin/perl

use warnings;      # Warnungen einschalten
use strict;        # Strict-Modus einschalten
use diagnostics;   # schaltet ausführliche Fehlermeldungen an

# Normales Programm fortführen

print "Hallo Welt!\n";
```

Perl-Applikationen: Packages und Module

* Für alles, was wiederverwendet werden soll
* Für alles spätestens ab ein paar hundert Zeilen Code
* Einfacheres Testing
* Tools für Build und Deployment
* Perl-Skripte nur noch für den Aufruf / Start der Applikation verwenden

Vorbereitung

Perl-Version und Entwicklungsumgebung

Aktuelles, eigenes Perl

* Für neue Projekte bietet sich Perl 5.10.x an
* Unter Linux/Unix (inkl. OS X) ein eigenes Perl installieren
  * kollidiert nicht mit dem System (CPAN-Module)
  * anpassbar (Compiler-Switches)
* Beispielsweise unter `/usr/local/perl/myperl`

Erdbeeren vor die Fenster

* Wer schon mit Windows gestraft ist, braucht sich nicht auch noch mit ActivePerl bestrafen!
* Strawberry-Perl bietet:
  * weniger Gebastel
  * volle CPAN-Integration
  * C-Compiler und so weiter – alles mit dabei

Entwicklungsumgebung

- Nutzen Sie eine Entwicklungsumgebung!
  - Kommandozeilenfreaks: vim oder emacs zur IDE aufrüsten
  - GUI-Umgebungen sind meist einfacher
    - Eclipse+EPIC (Perl Plugin), Komodo oder Padre
- Den meisten Editoren wie UltraEdit fehlen viele Funktionen wie Syntaxcheck, Perl::Tidy-Integration, …

CPAN-Tauglichkeit

Schreiben Sie Ihren Code so, als würde er aufs CPAN kommen

Keine Skripte schreiben!

- Schreiben Sie keine Skripte, außer zum Starten vom Code in Modulen (und für die Tests)
  - Wir haben:
    - Einzeiler
    - Skripte
    - Applikationen

Start-Skript

* Einfaches Beispiel-Skript – aller relevanter Code steckt in Modulen

```
#!/usr/bin/perl

use strict;
use warnings;

use Project::Tool::App;

Project::Tool::App->run(@ARGV);
```

Nutzen Sie *Module::Starter*

- Nutzen Sie *Module::Starter* oder ähnliches zum Anlegen von Projekten
  - Erstellt Grundgerüst für ganze Distributionen
    - *Module::Starter::Smart* erlaubt späteres hinzufügen neuer Module
    - *Module::Starter::PBP* erlaubt Templates
  - Ein eigenes *create-module.pl* kann hilfreich sein

Nutzen Sie *Module::Build*

- Nutze ein Build-System, beispielsweise *Module::Build*
  - Integration von Test, Code-Coverage-Testing und Packaging
  - Der eigene Code ist CPAN-Kompatibel
  - Neue Entwickler finden sich schnell zurecht

CPAN

Das CPAN gehört zu Perl dazu!

Nutzen Sie das CPAN

* Gibt es eine Lösung für mein Problem auf dem CPAN?
  * Das CPAN hat über 17 000 Distributionen und 65 000 Module.
  * Vieles ist ohne CPAN gar nicht möglich, DBI zum Beispiel
  * Nutzen Sie es!

System aktuell halten

* Halten Sie Perl und die CPAN-Module aktuell
* Zwei Philosophien:
  * Einmal installieren, dann *never touch a running system*
  * Oder: regelmäßig aktualisieren und aktuell halten
* Ersteres lässt sich oft nicht durchhalten
  * z.B. neue Module und Ärger mit anderen veralteten
* Daher in der Praxis häufiges aktualisieren meist besser

CPAN nicht nutzbar?

* Installation durch Firewall nicht möglich
  * Also: Modul manuell runterladen, Abhängigkeit feststellen, diese manuell runterladen, Abhängigkeiten feststellen, …
* Das ist vollkommen unpraktikabel
  * Und oft ein Grund, kein CPAN zu nutzen

Lokaler CPAN-Mirror

- Installieren Sie bei Bedarf einen lokalen CPAN-Mirror
- *CPAN::Mini* macht den Mirror einfach
  - Mirror nutzbar z.B. via *file://* oder *http://*
- Obacht mit Virenscannern: offizieller Test-Virus im ClamAV-Modul – Viren-Admin sollte den EICAR-„Virus“ kennen . . .

Moderne Techniken

Nutzen Sie moderne Perl-Techniken
Schauen Sie, was auf dem CPAN genutzt wird

Deployment mit PAR

* Installation aller genutzten und eigenen Module auf Zielsystemen kann aufwendig sein
* Sehr einfache Verteilung mit PAR möglich#!
* Etwas Aufwand bei weiteren Daten-Files

```
use PAR;
use lib qw(http://code.intern/non-prod/mein-code.par);
```

Tests und Staging

* Stages mittels PAR und Webserver oder gleich Subversion nachbilden
  * Stages Devel, Test, Non-Prod, Prod
    * Ab Test Code in PAR-Archiv packen und via HTTP verteilen
    * Nach Freigabe in die nächste Stage

```perl
#!/usr/bin/env perl

=head1 NAME

Beispiel-Applikation

=head1 BESCHREIBUNG

Diese Applikation lädt je nach Stage den passenden Code (Module) und führt diese dann aus

$ENV{REPOSITORY_URL} kann zum Beispiel sein: "http://code-server.local/svn";

=cut

use strict;
use warnings;

# Schon zur Compile-Time ausführen, daher BEGIN-Block
BEGIN
{
   if ( $ENV{REPOSITORY_URL} )
      {
      # Wenn der Code aus dem Subversion via PAR kommt:
      my $stage = $ENV{STAGE} || die "Environment-Variable STAGE ist nicht gesetzt!\n";
      my $fetch_path = "$ENV{REPOSITORY_URL}/par/$stage/project-tool.par";

      # Zur Laufzeit ausführen wegen der URL, daher eval
      eval q{
              use PAR;
              use lib '$fetch_path';
              return 1;
              } or die "PAR nicht installiert?\n $@";
      }
   else
      {
      # Wenn der Code von lokal kommen soll, Verzeichnisse einbinden
      eval q{
              use FindBin qw($Bin);
              use lib "$Bin/../lib";
              use lib "$Bin/../../Altlasten/lib";
              };
      }
} ## end BEGIN

use Project::Tool::App;                         # Applikation laden; kommt aus dem PAR-Archiv oder von lokal
exit Project::Tool::App->run(@ARGV);            # Applikation starten und deren Rückgabewert zurückgeben
```

Nutzen Sie Perls Möglichkeiten

* Nutze die Möglichkeiten, die Perl bietet
  * Nicht Perl wie C oder die Bash nutzen.
    * Perl hat Hashes, komplexe Datenstrukturen, Module, Reguläre Ausdrücke, Objektorientierung usw.
    * C-Style-Code ist schwer wartbar
  * Nicht die 100. Version von *join* oder den 1000. Kommandozeilen- oder Konfigurationsdateien-Parser bauen!

Nutzen Sie *Moose*

* *Moose* bietet ein modernes Objektsystem für Perl 5
  * Mit allem Schnickschnack
    * Einfach, komfortabel, mächtig
  * Basiert auf dem Perl 6 Objektsystem
  * Moose hat Attribute, Roles, Typ-Überprüfung, Methoden-Modifier, Delegation, . . .

Nutzen Sie *DBIx::Class*

* Bauen Sie keinen eigenen ORM – DBIx::Class nutzen
  * ORM-Wrapper sind hilfreich, um Datenbankabfragen objektorientiert zu kapseln
  * Das Handling kann praktisch sein, z.B. beim Weiterreichen im Code
  * Manuell ein ORM schreiben ist Wahnsinn
  * Komfort wird mit schlechterer Performance erkauft

Praktische Helfer wie *File::HomeDir*

* *File::HomeDir* ist praktisch, um systemunabhängig die korrekten Pfade für Dokumente, Einstellungen, Musik, Dokumente, … herauszufinden

* Manuelles Suchen keine gute Idee

  * Zum Beispiel ist nicht überall die Environment-Variable HOME vorhanden!

Temporäre Dateien

* Man kann sich prima unnötige Arbeit machen …
* => Legen Sie temporäre Dateien nicht manuell an!
* *File::Temp* legt temporäre Dateien und/oder Ordner sicher und zuverlässig an

```
my $dir = tempdir( CLEANUP => 1 );  # Dir. mit Auto-Aufräumer

my $file_handle = tempfile();       # temporäres File-Handle
```

Nutzen Sie *Log::Log4perl*

- Schreiben Sie Logs mit Log4perl
  - Ein *print* ist schnell hingeklatscht, aber nicht flexibel
  - Log4perl ist ein umfangreiches Logging-Framework
    - Verschiedene Log-Level
    - Verschiedene Log-Ziele wie Dateien, Screen, DBI, …
    - Sehr flexibel, gut erweiterbar

Schlusswort

Das Wort zum Ende