Ausgabe 1/2012 – www.foo-magazin.de

$foo

PERL MAGAZIN




Rex
Konfigurationsmanagement & Software-Deployment

Jenkins
Continuous Integration für Perl-Projekte

VMware
und Perl...

Nr 21

The people.
Leading technology.
Creating security.

astaro
Sophos Network Security

Andreas, Development Manager bei Astaro

Don’t panic. We are the good ones!

„Bei Astaro kann ich die Welt sicherer machen! Technologie und Kreativität sind gefragt, wenn wir neue Produkte entwickeln. Bald auch mit Dir?“

Warum Astaro? Weil Sie bei uns die Zukunft mitgestalten können!
Unser engagiertes Team entwickelt innovative Produkte in den Zukunftsmärkten IT-Sicherheit und OpenSource. Neben einem attraktiven Gehalt setzen wir auf flache Hierarchien, flexible Arbeitszeiten und einen offenen und respektvollen Umgang miteinander. Ihre Meinung zählt! Bringen Sie sich ein und werden Sie Teil eines motivierten und erfolgreichen Teams.

Astaro – Sophos Network Security
Gemeinsam sind Astaro und Sophos das größte Unternehmen Europas in der IT-Sicherheit. Wir bieten die erste Threat Management-Komplettlösung – zur Sicherheit unserer Kunden und zum nachhaltigen Erfolg unseres Unternehmens und unserer Mitarbeiter.

www.astaro.com/jobs

VORWORT

Renée Bäcker

100.000 EUR, ein neuer Pumpking und viel Aufmerksamkeit für Perl

Nach genau zwei Jahren hat Jesse Vincent im November 2011 seinen "Posten" als Pumpking aufgegeben. Er hat während dieser Zeit, die Perl-Entwicklung sehr stark belebt. Weniger durch eigenen Code - tatsächlich hat er kaum am Perl-Kern programmiert - als vielmehr durch Entscheidungen und Motivation.

So hat er auch seine Vision von einem zukünftigen Perl aufgezeigt. Nicht alles wird sich umsetzen lassen, aber es gab endlich mal nicht nur einen Rückblick, was sich über die Jahre getan hat, sondern auch einen Ausblick, was noch kommen wird.

Sein Nachfolger als Pumpking ist Ricardo Signes, der schon seit Jahren in der Perl-Community aktiv ist (z.B. mit Dist::Zilla). Ich wünsche Ricardo viel Spaß und viel Erfolg! Ich bin mir sicher, er wird seine Sache sehr gut machen.

Auch der Perl 5 Spendentopf hat Bewegung in die Perl-Entwicklung gebracht und im Dezember 2011 wurde dieser Spendentopf prall gefüllt: Booking.com hat 100.000 Euro an die Perl Foundation gespendet. Mit diesem Geld können ein paar gute Perl-Kernentwickler ein paar Monate bezahlt werden. Ich hoffe, dass auch weitere Unternehmen Geld geben werden - auch mit kleinen Beträgen kann man die Perl-Kernentwickler unterstützen.

Perl hat Ende Dezember 2011 auch viel Aufmerksamkeit erfahren - ohne eigenes großes Zutun: Auf dem Chaos Communication Congress 28C3 wurde in einem Vortrag gezeigt, dass viele Sprachen für DoS-Attacken anfällig sind. Bei dieser Schwachstelle geht es darum, dass die Sprachen viele Daten in Hashtabellen speichern und der Zugriff immer langsamer wird, wenn viele Einträge den gleichen Index haben. Die Perl-Entwickler hatten diese Schwachstelle schon 2003 behoben, indem sie die Berechnung des Index geändert haben. Nach diesem Vortrag auf dem 28C3 wurde Perl immer wieder als positives Beispiel genannt. Schön, wenn Perl einer so breiten Masse in einem so positiven Zusammenhang bekannt gemacht wird.

Natürlich bekommt Perl auch in dieser Ausgabe viel Aufmerksamkeit von uns und ich hoffe, dass wir wieder eine interessante Mischung an Artikeln zusammengestellt haben. Ich wünsche Ihnen viel Spaß beim Lesen der 21. Ausgabe des Perl-Magazins.

Die Codebeispiele können mit dem Code

9cx2K

von der Webseite www.foo-magazin.de heruntergeladen werden!

Alle weiterführenden Links werden auf del.icio.us gesammelt. Für diese Ausgabe: http://del.icio.us/foo_magazin/issue21

IMPRESSUM

Herausgeber:	Perl-Services.de Renée Bäcker Bergfeldstr. 23 D - 64560 Riedstadt
Redaktion:	Renée Bäcker
Lektorat:	Dominic Neumann
Anzeigen:	Renée Bäcker
Layout:	//SEIBERT/MEDIA
Auflage:	500 Exemplare
Druck:	powerdruck Druck- & VerlagsgesmbH Wienerstraße 116 A-2483 Ebreichsdorf
ISSN Print:	1864-7537
ISSN Online:	1864-7545
Kontakt:	info@perl-services.de

INHALTSVERZEICHNIS

ALLGEMEINES

ANWENDUNGEN

MODULE

NEWS

Hier werden kurz die Autoren vorgestellt, die zu dieser Ausgabe beigetragen haben.

Renée Bäcker

Seit 2002 begeisterter Perl-Programmierer und seit 2003 selbständig. Auf der Suche nach einem Perl-Magazin ist er nicht fündig geworden und hat so diese Zeitschrift herausgebracht. In der Perl-Community ist Renée recht aktiv - als Moderator bei Perl-Community.de, Organisator des kleinen Frankfurt Perl-Community Workshops, Grant Manager bei der TPF und bei der Perl-Marketing-Gruppe.

Herbert Breunung

Ein perlbegeisteter Programmierer aus dem ruhigen Osten, der eine Zeit lang auch Computervisualistik studiert hat, aber auch schon vorher ganz passabel programmieren konnte. Er ist vor allem geistigem Wissen, den schönen Künsten, sowie elektronischer und handgemachter Tanzmusik zugetan. Seit einigen Jahren schreibt er an Kephra, einem Texteditor in Perl. Er war auch am Aufbau der Wikipedia-Kategorie: "Programmiersprache Perl" beteiligt, versucht aber derzeit eher ein umfassendes Perl 6-Tutorial in diesem Stil zu schaffen.

Jan Gehring

Jan Gehring begeistert sich schon von klein auf für Computer und Programmieren. Seine 1en Gehversuche startete er mit einem Sinclair QL. Heute programmiert er leidenschaftlich gerne in Perl und ist für die Inovex GmbH als Linux Systems Architect unterwegs.

Thomas Fahle

Perl-Programmierer und Sysadmin seit 1996.

Websites:

- http://www.thomas-fahle.de
- http://Perl-Suchmaschine.de
- http://thomas-fahle.blogspot.com
- http://Perl-HowTo.de

Stefan Oberwahrenbrock

Stefan Oberwahrenbrock ist ausgebildeter Fachinformatiker mit dem Schwerpunkt Systemintegration. Seit gut 10 Jahren ist er für die TRANSDATA Soft- und Hardware GmbH tätig, die Logistiklösungen für die Transportwirtschaft entwickelt. Dort ist er mit der Administration der Serverlandschaft und der Clientsysteme betraut. Zu seinen Aufgaben zählen die Installation und Wartung von Web-, Datenbank- und Applikationsservern sowie von PC-Systemen. Er hat die plattformübergreifende Verfügbarkeit von Perl, den reichen Fundus des CPAN sowie die Perl-Community zu schätzen gelernt. Bei der Umsetzung diverser Projekte greift er gerne darauf zurück. Er ist Mitglied der Perl Mongers Bielefeld.

Stefan Oberwahrenbrock

VMware und Perl

Der Artikel beginnt mit einer zweckmäßig verkürzten Einführung in das Thema System-Virtualisierung im Allgemeinen und einer groben Übersicht über die Produktpalette des Herstellers VMware. Danach werden zwei für Perl-Programmierer interessante Software-Komponenten von VMware beschrieben. Es folgen Tipps und Erfahrungswerte zu deren Installation und Einrichtung. Zum Abschluss werden Code-Beispiele gezeigt, die als Einstieg und Basis für eigene Skripte verwendet werden können.

System-Virtualisierung

Bei der System-Virtualisierung werden die vorhandenen physischen Ressourcen eines Systems wie z. B. CPU, RAM und Festplatten so verfügbar gemacht, dass mehrere virtuelle Maschinen (VM) diese nutzen können. Ermöglicht wird dies durch eine Software-Virtualisierungsschicht, die als Hypervisor oder auch Virtual Machine Monitor (VMM) bezeichnet wird. Der Hypervisor fungiert als Vermittler und Koordinator, indem er die Systemaufrufe der virtuellen Systeme abfängt und an das Hardware-System weiterreicht. Eine virtuelle Maschine wird auch als Gast bezeichnet, die Kombination aus Hypervisor und Hardware-System als Host. In der einfachsten Ausprägung dieses Szenarios verwenden Gast und Host dieselbe Befehlssatzarchitektur. Hat ein Hypervisor beispielsweise eine x86-Architektur unter seiner Kontrolle, stellt er seinen Gästen ebenfalls eine x86-Architektur zur Verfügung. Man unterscheidet zwei Varianten von Hypervisoren: Die eine Variante wird direkt auf der Hardware installiert, die andere Variante benötigt ein installiertes Betriebssystem als Grundlage. In englischsprachiger Fachliteratur wird die erste Variante zuweilen als *native hypervisor* oder *bare metal hypervisor* bezeichnet, während man für die zweite Variante oftmals den Begriff *hosted hypervisor* vorfindet.

VMware

In dem aufgezeigten Virtualisierungsumfeld ist der im kalifornischen Palo Alto ansässige Softwarehersteller VMware seit mehreren Jahren tätig und verfügt mittlerweile über ein breit gefächertes Produkt-Portfolio in diesem Bereich. Eine grobe Unterteilung der aktuellen Produkte kann man z. B. anhand des verwendeten Hypervisor-Typs vornehmen. In den Bereich *bare metal hypervisor* fallen der *VMware vSphere Hypervisor (ESXi)* und sein Vorgänger, der *ESX-Server*. Diese Produkte müssen auf dedizierter Hardware installiert werden und stellen darüberhinaus gewisse Anforderungen an bestimmte Hardware-Komponenten wie z. B. CPU, Netzwerkkarten und das Festplatten-System (siehe Abbildung 1).

Abbildung 1: Bare metal hypervisor

Der *VMware vSphere Hypervisor* an sich ist mit seinen grundlegenden Virtualisierungsfunktionen kostenlos verfügbar. Kommen allerdings mehrere Server zum Einsatz, werden diese in der Regel durch zusätzliche, kostenpflichtige Management-Software in Form des *VMware vCenter Server* zentral verwaltet. Zudem können durch entsprechende Lizenzen erweiterte Funktionalitäten aktiviert werden. Das Konglomerat aus Servern und Management-Software ist in diversen Ausbaustufen mit unterschiedlichem Funktionsumfang erhältlich. Die *VMware vSphere* Produktschiene ist primär für den Einsatz im Unternehmensumfeld konzipiert.

Daneben gibt es mehrere Produkte, die dem Bereich *hosted hypervisor* angehören und auch im End- bzw. Heimanwenderbereich Verwendung finden. Hierzu zählen *VMware Player*, *VMware Workstation*, *VMware Fusion* (für Mac) und *VMware Server* (siehe Abbildung 2). Während *VMware Workstation* und *VMware Fusion* kommerzielle Produkte sind, kann man sich den *VMware Player* und den *VMware Server* kostenlos von der Web-Seite des Herstellers herunterladen. Hierzu ist allerdings eine Registrierung erforderlich. Die Weiterentwicklung des *VMware Server* wurde bereits vor einer ganzen Weile eingestellt. Die beiden kostenlosen Produkte weisen einen reduzierten Funktionsumfang auf. So kann der *VMware Server* beispielsweise lediglich einen Snapshot pro Gast erstellen, der *VMware Player* unterstützt diese Funktion gar nicht. Ein Snapshot ist ein gespeicherter Zustand des Gastes, den man bei Bedarf jederzeit wiederherstellen kann.

Abbildung 2: Hosted hypervisor

SDKs und APIs

Die einzelnen Produkte haben jeweils ihre eigene Benutzeroberfläche, mit der die verfügbaren Funktionen wie z. B. virtuelle Maschine erstellen, starten, stoppen, etc. bedient werden können. Darüber hinaus kann im Grunde jedes Produkt auch über eine API angesprochen werden, über die ebenfalls eine Vielzahl bzw. alle Funktionen des Produktes verwendet werden können. Auf der Webseite von VMware finden sich diverse Dokumentationen zu API und SDK und deren Verwendung [1]. Im Hinblick auf Perl sind dabei zwei Nennungen besonders interessant: Zum einen **VMware vSphere SDK for Perl** und zum anderen **VMware VIX API** - jeweils mit den darin enthaltenen Perl-Bindings.

Beim *VMware vSphere SDK for Perl* handelt es sich um ein clientseitiges Framework, das den Zugriff auf die *vSphere Web Services API* ermöglicht. Diese Services sind im wesentlichen Management-Funktionen wie z. B. virtuelle Maschine starten, stoppen, auf Snapshot zurücksetzen etc. In der aktuellen Version 5.0 sind allerdings auch erstmalig Funktionen integriert, die Aktionen innerhalb der virtuellen Maschine ermöglichen [2]. Diese umfassen Funktionen wie z. B. Dateien in die virtuelle Maschine kopieren, Programme starten, Prozesse beenden etc. Hierzu müssen in der virtuellen Maschine die *VMware Tools* installiert sein, die im jeweils vorliegenden Hypervisor-Produkt enthalten sind.

Die *VMware VIX API* ermöglicht schon länger sowohl die Ausführung von Management-Funktionen als auch von Aktionen innerhalb der virtuellen Maschine. Der Begriff *VIX* steht für *Virtual Infrastructure eXtension*. Die *VMware VIX API* kommt nicht nur in vSphere-Umgebungen zum Einsatz, sondern findet auch im Bereich der End- bzw. Heimanwender-Produkte Verwendung. Auch hier müssen in der virtuellen Maschine die *VMware Tools* installiert sein, um die API vollständig nutzen zu können. Es gibt Bindings für verschiedenen Sprachen, eine davon ist Perl.

In den derzeit aktuellen Versionen des *VMware vSphere*

SDK for Perl (5.0) und der *VMware VIX API* (1.11) haben sich Funktionsumfang und Einsatzzweck vermischt. Als Faustformel zur Orientierung kann man festhalten, dass das *VMware vSphere SDK for Perl* eher dem professionellen, kommerziellen Umfeld zuzuordnen ist, während die *VMware VIX API* auch im End- bzw. Heimanwenderbereich anzutreffen ist.

Zugang zu VMware-Downloads

Um die Produkte von VMware herunterladen zu können, muss man sich zunächst einen *VMware Account* erstellen. Hierzu hat der Hersteller die Webseite *http://www.vmware.com/account/login.do* eingerichtet. Im Bereich *New Customers* kann ein neuer Account angelegt werden. Möchte man sich als Privatperson einen Account erstellen, legt einem der Knowledgebase-Eintrag 1021642 folgendes Prozedere nahe: Im Feld *Company Name* möge man den Wert *Personal* eintragen und im Feld *Department* den Wert *Other* aus der angebotenen Liste wählen. Überflüssigerweise muss man dann trotzdem noch im Company-Bereich Werte eintragen, die einer syntaktischen Überprüfung standhalten. Möchte man in den Community-Foren Einträge posten und Blogs kommentieren, kann man weiter unter auf der Seite eine entsprechende Auswahl treffen. Anschließend erhält man eine E-Mail,, die einen Link zur Aktivierung des Account enthält. Nach Aufruf des Links und Eingabe des gewählten Passworts, steht der Account zur Verfügung. Mit diesem Account hat man nun auch Zugriff auf die Downloads. Pro Download muss man in der Regel noch einem *End User License Agreement* zustimmen. Dies ist auch für die freien Produkte nötig - der Vorgang wird auf den entsprechenden Web-Seiten auch als *Registration* bezeichnet.

Vorbereitungen für erste Tests

Möchte man API und/oder SDK einmal ausprobieren, kommt man verständlicherweise nicht um die Installation eines oder mehrerer VMware-Produkte herum. Sofern man bereits Zugang zu einem installierten Hypervisor aus dem Hause VMware hat, kann man direkt mit der Installation der API bzw. des SDK fortfahren. Andernfalls könnte man sich z. B. zunächst den *VMware Player* installieren, der für Windows und Linux zur Verfügung steht. Möchte man sich den Aufwand sparen, virtuelle Maschinen eigenhändig einzurichten, findet man z. B. auf der Web-Seite von VMware im Bereich *Virtual Appliances* fertig installierte Gast-Systeme zum Herunterladen. Aus lizenzrechtlichen Gründen sind dies Systeme mit einem frei verfügbaren Betriebssystem. Populäre Linux-Distributionen wie z. B. Mint, Fedora, Debian oder openSUSE sind dort in der Regel sogar in mehreren verschiedenen Versionen verfügbar. Wichtig ist, dass innerhalb der virtuellen Maschinen die besagten *VMware Tools* installiert sind. Für Windows und einige Linux-Distributionen sind im jeweils vorliegenden Hypervisor-Produkt binäre Installationspakete enthalten. Deren Installation im Gast-System ist recht komfortabel zu bewerkstelligen. Bei nicht direkt unterstützen Linux-Distributionen muss man oftmals innerhalb des Gast-Systems zunächst noch eine Entwicklungsumgebung installieren. Mit dieser kann das Setup der *VMware Tools* dann aus dem mitgelieferten Quellcode die Tools für das vorliegende Gast-Betriebssystem kompilieren und installieren.

Installation von SDK und API

Da die Installationspakete sowohl für Windows als auch für Linux verfügbar sind, können die Ausgangsbedingungen bei der Installation stark variieren. Einige Software-Konstellationen wurden für diesen Artikel getestet. Es folgen hierzu nun einige Erfahrungswerte.

VMware vSphere SDK for Perl 5.0

Sowohl unter Windows XP SP3 (32-Bit) als auch unter Windows 7 (64-Bit) verlief die Installation problemlos. Das Setup bringt ein ActivePerl in Version 5.8.8 mit, das von den mitgelieferten Skripten verwendet wird (siehe unten). Die Installation wurde nur auf Systemen getestet, auf denen noch kein Perl installiert war.

Unter Linux verlief die Installation leider nicht so reibungslos. Das Setup von VMware setzt offensichtlich ein Linux-System voraus, das mit einem RPM-basierten Paket-Manager arbeitet. Somit werden APT-basierte Systeme wie Debian und dessen Derivate nicht direkt unterstützt. Unter einem

openSUSE verlief die Installation erfolgreich, sowohl mit Version 11.4 als auch mit Version 12.1. Es müssen allerdings bestimmte Voraussetzungen erfüllt sein:

• Die C/C++-Entwicklungsumgebung muss vorhanden sein: RPM-Paket `patterns-openSUSE-devel_C_C++`

• Die "Library for WWW in Perl" muss vorhanden sein: RPM-Paket `perl-libwww-perl`

• Der Zugang zum CPAN muss eingerichtet sein und funktionieren: Die initiale Konfiguration muss vorhanden sein und die Internet-Verbindung muss bestehen, ggf. Test in einer CPAN-Shell mittels `reload index`.

• Die Umgebungsvariable PERL_LWP_SSL_VERIFY_HOSTNAME muss auf 0 gesetzt sein. Für den Zugriff auf einen vSphere-Management-Server via HTTPS muss die Umgebungsvariable PERL_LWP_SSL_VERIFY_HOSTNAME mit dem Wert 0 exportiert werden. Ist dies nicht der Fall, akzeptiert das zum Einsatz kommende `LWP::Protocol::https` das selbst signierte Zertifikat des Management-Servers nicht.

• Die Entwickler-Bibliotheken für XML2 müssen vorhanden sein (nur openSUSE 12.1). RPM-Pakete: `libxml2-devel-32bit libxml2-devel`

VMware VIX API 1.11

Sowohl unter Windows XP SP3 (32-Bit) als auch unter Windows 7 (64-Bit) verlief die Installation problemlos. Die Installation alleine reicht im Hinblick auf Perl allerdings nicht aus. Man muss zusätzlich noch die Perl-Bindings der API kompilieren und installieren. Im Installationsvereichnis findet sich hierzu die Datei *vix-perl.zip*. Entpackt man diese, findet sich in der darin enthaltenen README-Datei eine Anleitung zur Installation der Perl-Bindings. Laut der Anleitung werden zwei Dinge benötigt: Eine C-Entwicklungsumgebung und ein installiertes Perl. Die Anleitung legt einem nahe, ein ActivePerl in Version 5.10 sowie eine Entwicklungsumgebung von Microsoft zu verwenden. Die Erfahrung hat gezeigt, dass man in puncto Perl auch ein aktuelles ActivePerl 5.14 verwenden kann. Es muss sich allerdings um eine 32-Bit-Variante handeln, auch auf einem 64-Bit-System! Die Community-Edition von ActivePerl ist frei verfügbar [3]. Ältere Versionen stehen seitens ActiveState, dem Hersteller von ActivePerl, ohnehin nur für die kostenpflichtige Business-Edition zur Verfügung. Die Entwicklungsumgebung *Microsoft Visual Studio 2008 Express Editions mit SP1* wird von Microsoft kostenlos zum Download zur Verfügung gestellt [4]. Hierüber kann *Visual Studio C++ 2008 Express Edition* installiert werden. Die optionalen Produkte des Setups werden nicht benötigt. Nach erfolgreicher Installation startet man die *Visual Studio Eingabeaufforderung* (Start -> (Alle) Programme -> Microsoft Visual C++ 2008 Express Edition -> Visual Studio Tools -> Visual Studio 2008-Eingabeaufforderung). Auf Systemen mit aktivierter Benutzerkontensteuerung muss man dies mit **Administrator-Privilegien** tun, damit Dateien und Verzeichnisse innerhalb des Programm-Verzeichnisses erstellt werden können (Rechtsklick auf Visual Studio 2008-Eingabeaufforderung, aus Kontextmenü **Als Administrator ausführen** wählen). Anschließend wechselt man innerhalb der Eingabeaufforderung ins entpackte Verzeichnis der *VMware VIX API* (z. B. C:\Program Files (x86)\VMware\VMware VIX\vix-perl\) und schließt die Installation mit dem Dreigespann aus `perl Makefile.pl`, `nmake` und `nmake install` ab. Die auftretenden Warnungen scheinen der Funktionalität keinen Abbruch zu tun. Variationen der beteiligten Software-Komponenten wirkten sich nachteilig aus. So war es bei den Tests leider nicht möglich, die Installation mit einer anderen C-Entwicklungsumgebung (MinGW installiert durch ActivePerl) oder einem Strawberry Perl und dessen Build-Tools erfolgreich abzuschließen.

Unter Debian in Version 5 und 6 wurde die Installation zwar scheinbar erfolgreich (mit Warnungen) abgeschlossen. Allerdings war die API in Skripten nicht zu verwenden. Schon bei der grundlegenden Operation `HostConnect` erschien immer die Meldung `HostConnect() failed, 6000 The operation is not supported for the specified parameters`. Die Referenz der Fehlercodes brachte hierzu keinen weiteren Erkenntnisgewinn.

Mit einem openSUSE-System in Version 11.4 und Version 12.1 verlief die Installation erfolgreich, sofern eine C/C++-Entwicklungsumgebung vorhanden war (siehe oben). Nach erfolgreicher Installation befindet sich der Quellcode für die Perl-Bindings unter */usr/lib/vmware-vix/vix-perl.tar.gz*. Nach dem Entpacken des Archivs und dem Wechsel in das dabei erzeugte Verzeichnis */usr/lib/vmware-vix/vix-perl* kann man das Modul mit den drei Schritten `perl Makefile.PL`, `make`

und `make install` installieren. Die Warnungen während des `make` scheinen keine negativen Auswirkungen auf die Funktionalität zu haben. Es musste allerdings abschließend noch die Umgebungsvariable `LD_LIBRARY_PATH` um das Verzeichnis */usr/lib* erweitert werden. Hierzu reichte übergangsweise der Befehl `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/` in der aktuellen Terminal-Session aus.

Codebeispiele

VMware vSphere SDK

Das *VMware vSphere SDK for Perl* macht einem die ersten Schritte leicht. Es sind bereits fertige Perl-Skripte enthalten. Diese sogenannten *vSphere SDK for Perl Utility Applications* befinden sich unter Linux standardmäßig im Verzeichnis */usr/lib/vmware-vcli/apps/*, unter Windows im Unterverzeichnis *Perl\Apps* des Installationsverzeichnisses. Die einzelnen Skripte sind nach ihrer Funktion bzw. ihrem Einsatzgebiet in weitere Unterverzeichnisse unterteilt. So befinden sich im Verzeichnis *vm* Skripte, mit denen man Operationen für virtuelle Maschinen ausführen kann. Eines dieser Skripte ist `vmcontrol.pl`, mit dem man VMs starten oder stoppen kann. Ein Aufruf diese Skriptes sieht z. B. folgendermaßen aus:

```
vmcontrol.pl --username USER  \
   --password PASSWORD --operation poweron \
   --vmname "vm-debian6-x64-minimal" \
   --url \
https://mgmtsrv.your-net.local/sdk/webService
```

In diesem Beispiel wird ein zentraler *VMware vCenter Server* kontaktiert, über den die VM mit dem Namen "vm-debian6-x64-minimal" gestartet wird. Je nach Konfiguration gleicht der *VMware vCenter Server* die Authentifizierungsdaten mit lokalen Benutzern oder einem Active Directory ab. Zudem wird überprüft, ob der angegebene Benutzer überhaupt das Recht zur Durchführung der gewünschten Aktion hat.

Ist unter Linux die Umgebungsvariable `PERL_LWP_SSL_VERIFY_HOSTNAME` nicht auf 0 gesetzt (siehe oben), wird der Aufruf von `vmcontrol.pl` mit folgendem Fehler quittiert:

```
Server version unavailable at 'https://mgmtsrv.
your-net.local/sdk/vimService.wsdl'  at  /usr/
lib/perl5/5.14.2/VMware/VICommon.pm line 545.
```

Mit den vorgefertigten Skripten lassen sich bereits sehr viele Aktionen umsetzen. Wer darüber hinaus eigene Skripte erstellen möchte, findet im *vSphere SDK for Perl Programming Guide* eine Anleitung für den Einstieg dazu [5].

VMware VIX API

Die Online-Dokumentation der einzelnen Funktionen enthält auch Code-Beispiele, die im folgenden Skript verwendet werden (siehe Listing 1). Als Host-System und Ausführungsumgebung für das Skript kommt ein Windows 7 (64-Bit) zum Einsatz, auf dem der *VMware Player* in Version 4.0 installiert wurde. Die *VMware VIX API* und ActivePerl sind wie oben beschrieben installiert worden. Eine virtuelle Maschine mit openSUSE 12.1 als Gast-Betriebssystem wurde erstellt und mit den *VMware Tools* ausgestattet.

Die grundlegenden Schritte sind folgende: Nachdem die notwendigen VMware Module eingebunden worden sind, wird durch die Funktion `HostConnect` eine Verbindung zum Hypervisor aufgebaut. Hierbei muss der korrekte Typ des Hypervisors angegeben werden, der in diesem Fall durch den Wert von `VIX_SERVICEPROVIDER_VMWARE_PLAYER` spezifiziert wird. Anschließend wird ein Handle für die virtuelle Maschine erzeugt, wobei der Pfad zur zentralen Konfigurationsdatei der virtuellen Maschine angegeben werden muss (VMX-Datei). Die Funktion `VMPowerOn` fährt die Maschine hoch. Als Option wird hierbei `VIX_VMPOWEROP_LAUNCH_GUI` angegeben, wodurch eine neue Instanz des *VMware Players* erzeugt wird und man im zugehörigen Fenster den Bootvorgang beobachten kann. Durch den anschließenden Aufruf von `VMWaitForToolsInGuest` wartet das Skript bis zu 300 Sekunden darauf, dass die *VMware Tools* im Gast-System aktiviert sind. Da Aktionen innerhalb der virtuellen Maschine zur Ausführung die *VMware Tools* benötigen, wäre es zwecklos, ohne deren korrekten Start fortzufahren. Bevor die gewünschten Aktionen innerhalb des Gastes ausgeführt werden können, muss ein Authentifizierungskontext erzeugt werden. Hierzu werden der Funktion `VMLoginInGuest` ein Benutzername und das dazugehörige Passwort übergeben. Die nachfolgenden Aktionen unterliegen den Restriktionen des Gast-Betriebssystems, die für den angegebenen Benutzer gelten. Möchte man z. B. den Inhalt eines Verzeichnisses ausgeben, muss der Benutzer (hier *vmware*) auch die entsprechenden Leserechte im Dateisystem haben. Mit den abschließenden Aufrufen von `VMLogoutFromGuest`, `ReleaseHandle` und `HostDisconnect` wird die Session beendet.

```
use warnings;
use strict;

use VMware::Vix::Simple;
use VMware::Vix::API::Constants;

my $err = VIX_OK;
my $hostHandle = VIX_INVALID_HANDLE;
my $vmHandle = VIX_INVALID_HANDLE;

($err, $hostHandle) = HostConnect(VIX_API_VERSION,
                                  VIX_SERVICEPROVIDER_VMWARE_PLAYER,
                                  undef, # hostName
                                  0, # hostPort
                                  undef, # userName
                                  undef, # password
                                  0, # options
                                  VIX_INVALID_HANDLE); # propertyListHandle

die "HostConnect() failed, $err ", GetErrorText($err), "\n" if $err != VIX_OK;

($err, $vmHandle) = VMOpen($hostHandle,
                     "D:\\vm\\vm-opensuse12.1-x64-minimal\\vm-opensuse12.1-x64-minimal.vmx");
die "VMOpen() failed, $err ", GetErrorText($err), "\n" if $err != VIX_OK;

$err = VMPowerOn($vmHandle,
                 VIX_VMPOWEROP_LAUNCH_GUI, # powerOnOptions
                 VIX_INVALID_HANDLE);  # propertyListHandle
die "VMPowerOn() failed, $err ", GetErrorText($err), "\n" if $err != VIX_OK;

$err = VMWaitForToolsInGuest($vmHandle,
                             300); # timeoutInSeconds
die "VMWaitForToolsInGuest() failed, $err ", GetErrorText($err), "\n" if $err != VIX_OK;


$err = VMLoginInGuest($vmHandle,
                      "vmware", # userName
                      "vmware", # password
                      0); # options
die "VMLoginInGuest() failed, $err ", GetErrorText($err), "\n" if $err != VIX_OK;

# --- add your actions here ---

$err = VMLogoutFromGuest($vmHandle);
die "VMLogoutFromGuest() failed, $err ", GetErrorText($err), "\n" if $err != VIX_OK;

ReleaseHandle($vmHandle);
HostDisconnect($hostHandle);
```

Listing 1

Im Platzhalter-Bereich des Beispiels können die eigenen Aktionen hinzugefügt werden. Welche Funktionen es gibt und wie sie zu verwenden sind, ist in der Online-Dokumentation der API aufgeführt [6]. Es empfiehlt sich hier, die Referenz sortiert nach Funktionen für Perl aufzurufen. Als abschließendes Beispiel sei hier noch die Funktion `VMListDirectoryInGuest` aufgeführt, mit der man auch die Auswirkungen des Authentifizierungskontextes zeigen kann:

```
my @fsObjectRefs;
($err, @fsObjectRefs) =
    VMListDirectoryInGuest($vmHandle,
                           '/root', 0);
die "VMListDirectoryInGuest() failed, $err ",
  GetErrorText($err), "\n" if $err != VIX_OK;

foreach my $ref (@fsObjectRefs) {
   print "file: $ref->{'FILE_NAME'}\n";
}
```

Diese Aktion schlägt aufgrund von fehlenden Leserechten des Benutzers *vmware* für das Verzeichnis */root* fehl:

```
 VMListDirectoryInGuest() failed, 13 You do
not have access rights to this file
```

Ändert man das Verzeichnis z. B. auf */tmp* ab, kann die Aktion erfolgreich ausgeführt werden:

```
file: .ICE-unix
file: vmware-root
file: .XIM-unix
file: .font-unix
file: .X11-unix
file: VMwareDnD
file: vmware-config0
file: vmware-file-mod0
file: .Test-unix
```

Fazit

Schade ist, dass sich sowohl *VMware VIX API* als auch *VMware vSphere SDK for Perl* nur unter bestimmten Betriebssystemen und in bestimmten Softwarekonstellationen installieren bzw. verwenden lassen. Hier wäre etwas mehr Plattformunabhängigkeit wünschenswert. Kann man diesen Umstand aber in Kauf nehmen und sich auf die funktionierenden Zusammensetzungen beschränken, gibt einem der Hersteller VMware recht mächtige Werkzeuge mit vielfältigen Einsatzmöglichkeiten an die Hand. So können API und SDK z. B. im Rahmen von Softwaretests nützlich sein, bei denen sich wiederholende Abläufe automatisiert werden sollen. Auch bei der Erstellung von Plugins für Monitoringlösungen wie *Nagios/Icinga* kann man auf sie zurückgreifen. Ebenso können sie eine Hilfe bei der Administration einer virtuellen Umgebung sein. Leute, die ohnehin mit Perl arbeiten, können sich durch API und SDK auf vertrautem Terrain neue Möglichkeiten erschließen.

Links

[1] VMware API and SDK Documentation: *http://www.vmware.com/support/pubs/sdk_pubs.html*

[2] VIX Guest Operations functionality in core vSphere 5 APIs: *http://blogs.vmware.com/vix/2011/07/vix-guest-operations-functionality-in-core-vsphere-5-apis.html*

[3] ActivePerl Community Editon: *http://www.activestate.com/activeperl/downloads*

[4] Microsoft Visual Studio 2008 Express Editions mit SP1: *http://www.microsoft.com/downloads/de-de/details.aspx?FamilyID=3254C868-BCB9-412C-95C6-D100C872EC60*

[5] vSphere SDK for Perl Programming Guide: *http://www.vmware.com/support/developer/viperltoolkit/*

[6] VIX API Reference 1.11: *http://www.vmware.com/support/developer/vix-api/vix111_reference/*

YAPC::EU 2012
August 2012 in Frankfurt

Die YAPC::EU ist mit rund 300 Teilnehmern die größte europäische Veranstaltung rund um Perl...

Interesse an Sponsoring?
sponsoring@yapc2012.de

Renée Bäcker

Continuous Integration für Perl-Projekte mit Jenkins

Der Termin für die Abgabe des Projekts rückt immer näher und ein paar wichtige Features fehlen noch. In mehreren Nachtschichten wird noch jede Menge Code in das Repository eingespielt. Für Tests bleibt jetzt kaum noch Zeit. Als der Kunde einen Blick auf die Anwendung wirft, fallen ihm gleich ein paar Fehler auf. Nacharbeiten werden notwendig, auf Seiteneffekte wird trotzdem kaum geachtet.

Diese Situation zeigt, dass Tests nicht vernachlässigt werden sollten. Vielleicht auch noch ein Spruch von einem meiner Mitarbeiter: *"Habe mich heute in die Test-Suite von Perl eingearbeitet und auch gleich Tests für meinen Code geschrieben [...] Das hat tatsächlich sehr geholfen... habe eine Unmenge an "Bugs" entdeckt. Schon sehr Sinnvoll die ganze Sache."*

Aber gerade dann wenn es hektisch wird, werden Tests außer Acht gelassen. Und wenn die Entwicklung im Team vorgenommen wird, ist das Testen noch wichtiger. Wie vertragen sich meine Änderungen mit den Änderungen der Kollegin? Perl bietet auf dem CPAN so viele Module, die beim Schreiben von Tests große Unterstützung bieten. Zwei davon werden auch bei den CPAN-News in dieser Ausgabe ganz kurz vorgestellt. Mit *prove* bzw. `make test` ist der Testdurchlauf auch schnell gestartet.

Aber man denkt nicht immer daran, die Tests zu starten oder man ist der Meinung, dass die eigenen Änderungen ja nur winzig sind und sich auf die Funktionalität nicht wesentlich auswirken. Aus diesem Grund sollte man einen Diener haben, der die lästige Aufgabe des Testens übernimmt.

Ein solcher Diener/Butler ist Jenkins, ein *Continuous Integration* (CI) Server, der aus dem CI-Server Hudson entstanden ist. Dieser Artikel stellt Continuous Integration vor und zeigt, wie Jenkins dazu verwendet werden kann, die Qualität der eigenen Projekte zu steigern.

Was ist Continuous Integration?

Mit Integration ist das Zusammenbauen der einzelnen Programmkomponenten zur Gesamtanwendung gemeint. In den meisten Projekten wird das Bauen der Gesamtanwendung erst kurz vorm Release gemacht, so dass Fehler im Prozess erst kurz vor wichtigen Terminen auffallen. Werden Programmkomponenten regelmäßig, also kontinuierlich, zusammengeführt, fallen die Fehler früher auf.

In der Softwareentwicklung ist es so, dass die Kosten für die Beseitigung von Fehlern enorm ansteigen, je später im Prozess die Fehler gefunden werden. Zusätzlich wird die Fehlersuche wesentlich vereinfacht, je früher der Fehler entdeckt wird, da weniger Code-Änderungen nach dem Fehler durchsucht werden müssen. Damit sinkt auch der Integrationsaufwand am Ende des Projekts. Sollten dann noch Fehler auftreten, sind die Nacharbeiten schneller erledigt.

Die kontinuierliche Integration ist also ein Mittel, um die Risiken zu minimieren und die Qualität der Software zu steigern. Bevor der Prozess automatisiert werden kann, muss genau geplant werden, wie dieser Prozess aussieht. Nach der Planung und Formalisierung des Prozesses, ist es auch für Einsteiger im Projekt einfach, "mal eben" die Software zu bauen. Und auch für die alten Hasen im Projekt sinkt die Angst, einen wichtigen Schritt in der Integration zu vergessen.

Im Laufe eines Integrationslaufes greift der CI-Server auf verschiedene andere IT-Systeme zu: Versionskontrollsysteme, Build-Systeme und weitere.

Kontinuierlich bedeutet hier entweder täglich oder sogar bei jedem Commit. Im Laufe des Artikels werde ich zeigen, wie beides bei Perl-Projekten umgesetzt werden kann.

Der Ablauf während der Entwicklung sieht dann folgendermaßen aus: Die Entwickler programmieren auf Ihren eigenen Rechnern. Die Entwicklungen landen dann im Versionskontrollsystem. Der CI-Server wird vom Versionskontrollsystem angetriggert und darüber informiert, dass Änderungen vorliegen. Der CI-Server stößt den Build-Prozess an. Nach dem Integrationslauf werden die Ergebnisse auf dem CI-Server zur Verfügung gestellt, sehr häufig werden die Ergebnisse auch über weitere Kanäle an die Entwickler verteilt, z. B. per E-Mail.

Aber bevor es losgehen kann, muss Jenkins/Hudson installiert werden. Jenkins ist eine Java-Webanwendung und benötigt die *Java Runtime Environment* (JRE) mindestens in Version 1.5. Unter Ubuntu/Debian ist die Installation ganz einfach, da es fertige Pakete gibt, die über die Paketverwaltung eingespielt werden können.

Konfiguration von Jenkins

Nach der Installation ist Jenkins schon direkt lauffähig. Einfach im Browser *http://localhost:8080/* aufrufen und schon ist man drin.

Da wir Jenkins hier nur für Perl-Projekte benutzen wollen, ist gar nicht so viel zu konfigurieren. Da interessieren uns weder Maven noch die Version des Java Development Kits. Da wir aber die Ergebnisse der Integrationsläufe auf unseren Schreibtisch bekommen wollen, richten wir den E-Mail-Versand ein.

Gerade wenn man den CI-Server nicht nur für sich selbst laufen lässt, sondern für mehrere Leute, sollte man vielleicht auch noch einen anderen Servernamen vergeben und im DNS eintragen.

Das wars aber auch schon. Jetzt kann der CI-Server benutzt werden.

Ein einfaches Perl-Projekt/-Modul

Als Beispiel für die kontinuierliche Integration mit einem Perl-Projekt soll ein einfaches Perl-Modul herhalten. Der Aufbau ist in der folgenden Baumstruktur dargestellt.

```
jenkins@ubuntu:~/MyJenkins$ tree
.
├── lib
│   └── MyJenkins.pm
├── Makefile.PL
├── MANIFEST
└── t
    ├── 00_load.t
    ├── 01_basic.t
    └── 02_output.t
```

Mit dieser Struktur hält sich das Modul an den inoffiziellen Standard von CPAN-Modulen: Das Modul im *lib/*-Verzeichnis, das Skript zum Generieren des Makefiles und die MANIFEST-Datei im Hauptverzeichnis und ein Verzeichnis *t/* mit den Tests.

Im Test-Verzeichnis sind drei Dateien abgelegt, mit der die Funktionen des Moduls getestet werden. So wie das Modul hier dargestellt ist, liegt es in einem SVN-Repository. Die Arbeit mit SVN-Repositories ist bei Jenkins standardmäßig möglich. Andere Versionsverwaltungssysteme werden nur über Plugins unterstützt. Wie das mit Git geht, wird in späteren Abschnitten noch gezeigt.

Der übliche Integrationsprozess sieht bei Perl-Modulen ja mit dem folgenden Vierzeiler überall ähnlich aus:

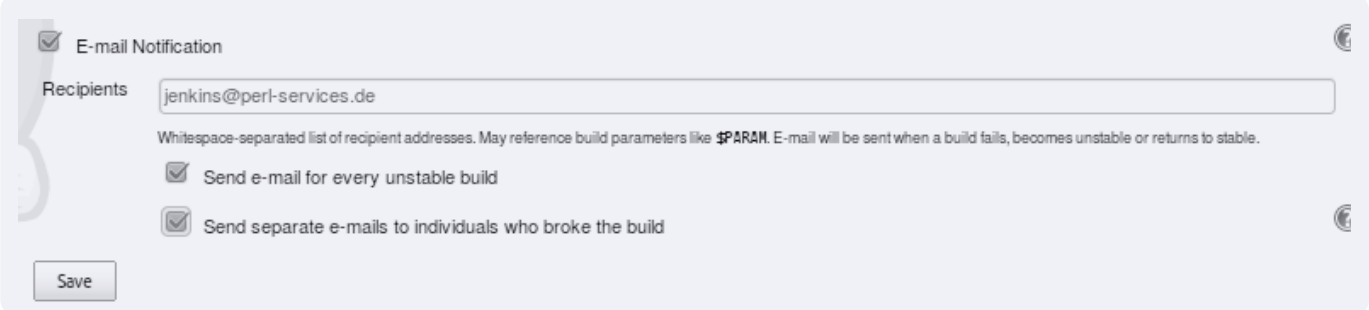


Abbildung 1: Konfiguration des E-Mail-Versands

```
perl Makefile.PL
make
make test
make dist
```

Weitere `make`-Befehle können auch noch enthalten sein, z.B. für die Erstellung der *MANIFEST*-Datei. Bevor das Modul gebaut werden kann, muss es noch aus dem SVN-Repository exportiert werden. Das geht mit dem Befehl

```
svn export /tmp/Modul/
```

Mit diesen Erkenntnissen haben wir den Build-Prozess formalisiert. Wie oben schon erwähnt ist das der erste Schritt zur Nutzung von Continuous Integration. Nachdem der erste Schritt getan ist, kann das Projekt in Jenkins eingetragen und konfiguriert werden.

Der erste Job

Solche einzelnen Aufträge für Jenkins werden als *Job* bezeichnet. Der Job besteht aus dem Beziehen des Quellcodes aus der Versionsverwaltung, dem Testen, erstellen des Pakets und eventuell noch weiteren Schritten wie dem Deployment.

Jenkins kennt auch unterschiedliche Jobtypen: "Free Style"-Projekte, Maven-Projekte und Multikonfigurationsprojekte. Maven-Projekte und Multikonfigurationsprojekte interessieren uns an dieser Stelle nicht. Für unsere Perl-Projekte erstellen wir "Free Style"-Jobs. Vielleicht sollte mal über Jenkins-Plugins für Perl-Projekte nachgedacht werden (siehe Abbildung 2).

"Free Style"-Projekte erlauben eine ziemlich große Freiheit bei der Konfiguration und die brauchen wir auch. In unserem ersten Beispiel liegt der Code in einem SVN-Repository, also

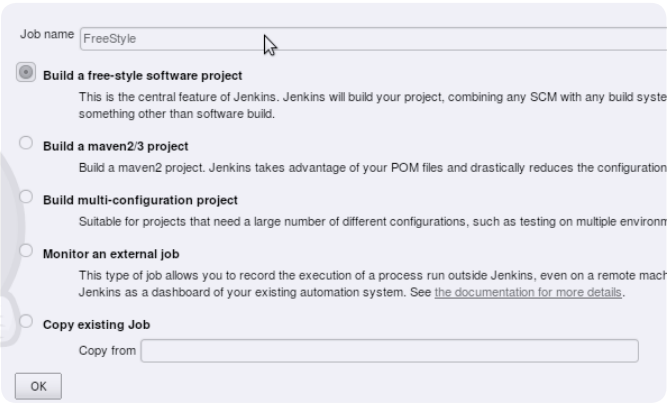


Abbildung 2: Einstieg in die Job-Konfiguration: ein "Free Style"-Projekt

wählen wir die entsprechende Box und geben die notwendigen Daten ein (Abbildung 3). In diesem Fall benutzen wir immer einen `checkout`, um immer auf einer ganz sauberen Basis zu arbeiten.

Hierbei ist dann aber zu beachten, dass ein `checkout` natürlich länger braucht, als wenn man immer nur ein `update` macht. Welche Strategie man fährt ist immer eine Entscheidung, die von Fall zu Fall anders aussehen kann.

Jetzt kommen die Überlegungen dazu zum Tragen, wie ein Build in unserem Projekt funktioniert. Denn als nächstes muss das Build-System eingestellt werden. In unserem Fall arbeiten wir mit `ExtUtils::MakeMaker` und Makefiles. Mit anderen Build-Systemen wie z. B. `Module::Build` funktioniert das Ganze aber analog.

Wir müssen also auf der Shell den ersten Befehl ausführen (Abbildung 4).

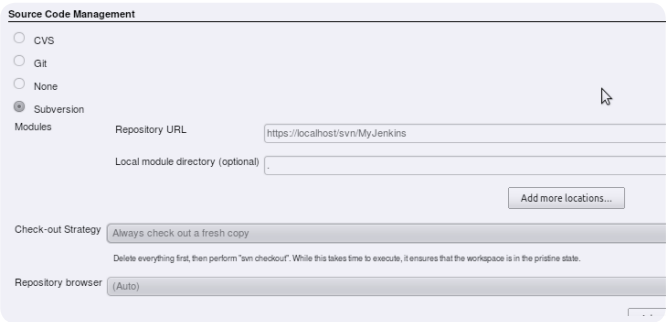


Abbildung 3: Der Quellcode wird mit SVN verwaltet

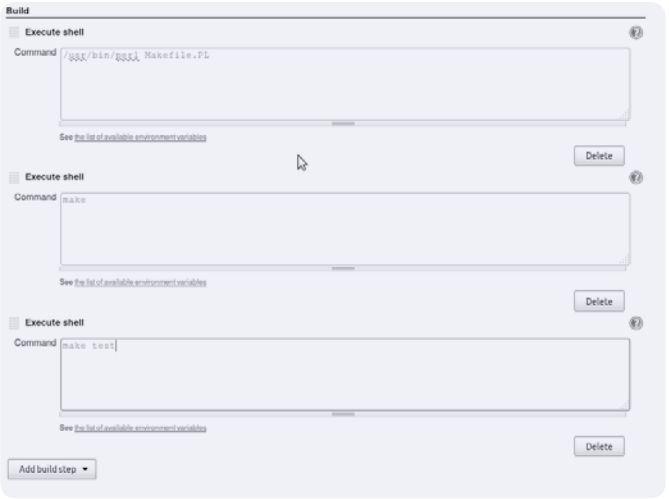


Abbildung. 4: Konfiguration des Build-Prozesses

Danach geht der Build-Prozess erst richtig los! Für die beiden anderen Befehle werden auch Shell-Befehle eingetragen.

Nachdem die Konfiguration gespeichert ist, können wir den ersten Integrationslauf starten. In Jenkins werden die letzten Build-Läufe angezeigt und man kann sich anschauen, was auf der Konsole ausgegeben wurde. Nicht wundern über den Blauen Punkt, es ist alles in Ordnung. Statt der üblichen Ampelfarben Rot, Gelb und Grün gibt es bei Jenkins die Farben Rot, Gelb und Blau. Das hat angeblich mehrere Gründe: Zum einen ist es für Personen mit Rot-Grün-Schwäche einfacher den tatsächlichen Status zu erkennen, der andere Grund ist eine sprachliche Feinheit im Japanischen (der Autor von Jenkins ist Japaner).

TAP vs. JUnit

Wie bei Perl-Modulen üblich, ist die Ausgabe nach dem *Test Anything Protocol* (TAP) formatiert, da Jenkins aber aus der Java-Welt kommt und kein TAP versteht, muss das TAP durch den Java-Quasistandard *JUnit* ersetzt werden. Der Build-Prozess wurde zwar als erfolgreich markiert, aber die Test-Ergebnisse werden nicht aufbereitet dargestellt.

Aber wir sind nicht die ersten mit diesem Problem: Auf CPAN gibt es wie (fast) immer ein fertiges Modul, mit dem wir uns viel Arbeit ersparen können. Als erstes benutzen wir statt `make test` einfach das Tool *prove*. Bei diesem Tool kann man auch eigene *Formatter* angeben und für JUnit gibt es schon einen fertigen Formatter auf CPAN: *TAP::Harness::JUnit*.

```
prove -v --harness=TAP::Harness::JUnit
```

Genau das kann jetzt auch in der Job-Konfiguration eingetragen werden. Zusätzlich stellen wir jetzt noch ein, dass die Ergebnisse veröffentlicht werden sollen (Abbildung 5). *prove* erstellt die Datei *junit_output.xml*, die Jenkins als Quelle für die Veröffentlichung dient.

Abbildung. 5: Übersicht der Tests nach einem Integrationslauf

Speicherung der Daten

Jenkins speichert alle Daten der Integrationsläufe im Dateisystem unterhalb des Verzeichnisses, das in der Umgebungsvariablen `JENKINS_HOME` steht. Der Verzeichnisbaum sieht so aus (gekürzt):

```
.
├── ...
├── jobs
│   └── MyJenkins - Simple Example
│       ├── builds
│       │   ├── 1 -> 2011-12-18_02-41-00
│       │   ├── 2011-12-18_02-41-00
│       │   │   ├── build.xml
│       │   │   ├── changelog.xml
│       │   │   └── log
│       ├── config.xml
│       ├── lastStable
│       ├── lastSuccessful
│       ├── nextBuildNumber
│       ├── subversion.credentials
│       └── svnexternals.txt
├── nodeMonitors.xml
├── plugins
├── updates
├── userContent
└── workspace
    └── MyJenkins - Simple Example
```

Im *jobs*-Verzeichnis liegen dann alle notwendigen Informationen zu dem einzelnen Job und dessen Integrationsläufen. Die Informationen, die in Jenkins angezeigt werden, sind alle hier gespeichert. Die eigentliche Arbeit passiert im Verzeichnis *workspace*. Dorthin werden die Quellen ausgecheckt und der Build-Prozess ausgeführt.

Module mit Dist::Zilla und mit Git verwalten

Bisher war es ein ganz einfaches Beispiel und der Jenkins-Job läuft ohne Probleme. Mittlerweile hat sich aber `Dist::Zilla` immer mehr zur Verwaltung von Perl-Modulen durchgesetzt. Mit `Dist::Zilla` muss man sich nicht mehr selbst um die Erstellung von `Makefile.PL` kümmern, das Erstellen der Distributionen ist einfacher und man kann mit einem einzigen Befehl das Perl-Modul bauen und auf CPAN laden. `Dist::Zilla` ist sehr flexibel und bietet extrem viele Möglichkeiten. Es wird in einer der kommenden Ausgaben von `$foo` näher vorgestellt.

An dieser Stelle nur so viel, dass mit dem oben gezeigten Vierzeiler nichts zu holen ist. Man muss jetzt erst mit einem

`dzil build` die Dateien und Strukturen erschaffen, damit man in der Situation ist wie bei dem einfachen Beispiel von Anfang an. Es sieht dann also so aus:

```
dzil build
cd MyJenkins-<Version>
perl Makefile.PL
make
prove -v --harness=TAP::Harness::JUnit
```

Ein Problem ist jetzt, dass wir nicht automatisch wissen, in welcher Version das Modul gerade vorliegt und somit kennen wir den Verzeichnisnamen nicht. Deshalb müssen wir beim `dzil build` noch einen Parameter mitgeben:

```
dzil build --in MyJenkinsBuild
```

Das muss jetzt entsprechend in die Job-Konfiguration eingetragen werden (Abbildungen 6).

Da der Build in einem Unterverzeichnis (`MyJenkinsBuild`) stattfindet, ist dort auch die JUnit-Datei zu finden. Deshalb muss hier noch der Verzeichnisname angegeben werden. Und noch eine Besonderheit fällt hier sicherlich auf: Bei den Verzeichnisnamen wurde hier noch die Umgebungsvariable `BUILD_NUMBER` verwendet. Bei den Praxistests hat sich gezeigt, dass es Probleme gab, wenn der Build immer in `MyJenkinsBuild` passiert. Die Integrationsläufe schlugen immer fehl. Das hängt damit zusammen, dass immer Artefakte übrigbleiben.

Wenn die `BUILD_NUMBER` noch mit aufgenommen wird, wird bei jedem Durchgang ein neues Verzeichnis ohne Artefakte erzeugt und der Durchgang wird erfolgreich abgeschlossen, wenn kein Fehler in den Tests auftaucht.

Es gibt jetzt aber noch einen weiteren Unterschied zu dem einfachen Beispiel: Der Code liegt nicht in einem SVN-Repository, sondern wird mit Git verwaltet. Damit kann Jenkins standardmäßig nicht umgehen. Aber durch die Möglichkeit, Jenkins mittels Plugins zu erweitern, ist das kein Problem. Es gibt ein Plugin, damit Jenkins mit Git umgehen kann. Das Plugin ist im Plugin-Verzeichnis auf der Jenkins-Seite zu finden.

Zur Installation des Plugins gehen wir einfach in der Jenkins-Installation im Konfigurationsmenü auf *Plugin Manager*. In der Liste der verfügbaren Plugins findet man für sehr viele Anwendungsfälle schon ein fertiges Plugin. Und man erkennt, dass es Plugins für verschieden Kategorien/Funktionen gibt. Z. B. für die Benutzerauthentifizierung oder eben für weitere Versionsverwaltungssysteme. Wir wählen das "Git Plugin" aus und installieren es.

Nachdem das Plugin installiert ist, können wir auch entsprechend Git als Quelle für den Code konfigurieren (Abbildung 7).

Es ist dabei darauf zu achten, dass der Benutzer unter dem Jenkins läuft einen SSH-Key ohne Passwort hat und dieser SSH-Key in der Konfiguration von *gitosis* oder *gitolite* eingetragen ist. Das Git-Plugin kann nämlich nicht damit umgehen, wenn der Zugriff mit einem Passwort geschützt ist. Dann kann Jenkins das Git-Repository nicht klonen und der Build-Prozess schlägt fehl. Es können auch einzelne Branches geklont werden. Das ist ganz praktisch, um auch parallele Entwicklungen frühzeitig testen zu können.

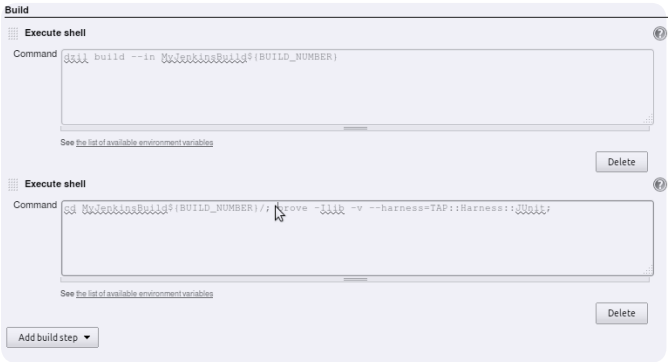


Abbildung 6: Der Build-Prozess bei einem Modul mit Dist::Zilla

Abbildung 7: Git als Code-Quelle

Ausführen der Integrationsdurchläufe

Jetzt sind die Projekte in Jenkins eingetragen. Der Diener weiß also, was er zu tun hat, wenn er gerufen wird. Jetzt müssen die Integrationsdurchläufe nur noch gestartet werden. Wie schon eingangs erwähnt, kann man das auf mehreren Wegen erreichen. Zum einen manuell, zum anderen über einen regelmäßigen Dienst (unter Linux als cronjob, unter Windows als Task). Eine weitere Möglichkeiten stellen Hooks im Versionskontrollsystem (VCS) dar. Diese verschiedenen Möglichkeiten werden in den folgenden Abschnitten gezeigt und erläutert.

Manuelles Starten der Integration

Gerade wenn ein Job neu eingerichtet ist, möchte man ja ungern bis zum nächsten Morgen warten, um das Ergebnis der Integration zu sehen. Oder ob die Konfiguration überhaupt richtig war. Deshalb kann man einen Integrationslauf auch einfach manuell starten. Auf der Job-Detailseite in Jenkins gibt es links im Menü den Link *Build Now*. Über den wird der nächste Durchgang in eine Queue geschoben und dann abgearbeitet.

Cronjob / Task

Schon bei der Konfiguration des Jobs kann man unter *Build Trigger* einstellen, ob der Build-Durchgang periodisch vorgenommen werden soll. Die Syntax ist hier wie bei Cronjobs, nur dass kein Programm aufgerufen wird. Hier werden nur die Zeiten eingestellt. So wird zum Beispiel immer nachts um 1:10 Uhr ein Integrationslauf gestartet:

```
10 1 * * *
```

Abfrage der Versionsverwaltung

Ähnlich wie bei den regelmäßigen Build-Läufen kann auch die Versionsverwaltung regelmäßig abgefragt werden. Immer wenn eine Änderung im Repository vorliegt, wird ein neuer Build-Lauf angestoßen. Um dieses Feature zu nutzen muss bei der Konfiguration des Jobs *Poll SCM* gewählt werden und in Cronjob-Syntax die Zeitabstände eingetragen werden.

Hooks

Um nicht immer wieder beim VCS nachzufragen, ob es Änderungen gibt und daraufhin einen neuen Build-Lauf zu starten, kann man auch den umgekehrten Weg gehen und Jenkins informieren, sobald ein Commit gemacht wird.

Beispielhaft soll hier die Integration der Hooks mit SVN und Git gezeigt werden. Aber das Prinzip der Hooks ist in den meisten VCS vorhanden und das Übertragen der Beispiele von hier auf andere Systeme sollte kein großes Problem darstellen.

Beim Git-Plugin ab Version 1.1.14 ist es ganz einfach. Allein durch den Aufruf einer URL können alle Jobs benachrichtigt werden, die auf ein bestimmtes Git-Repository zugreifen:

```
http://localhost/git/
  notifyCommit?url=<repo_url>
```

Das kann jetzt in ein Git-Hook eingebaut werden. Wenn es aus dem zentralen Git-Repository heraus geschehen soll, in das alle Entwickler `push`en, empfiehlt sich ein `post-receive`-Hook.

Dort wird einfach

```
exec curl
     http://localhost:8080/git/
     notifyCommit?url=
      jenkins@localhost:MyJenkinsDzil.git
```

eingetragen.

Allerdings ist es hierbei notwendig, dass das Polling wie im vorigen Abschnitt beschrieben konfiguriert ist.

Natürlich kann man auch über die normale API einen Build anstoßen. Allerdings müssen hier die Zugangsdaten über-

```
REPOS="$1"
REV="$2"
UUID=`svnlook uuid $REPOS`
/usr/bin/wget \
  --header "Content-Type:text/plain;charset=UTF-8" \
  --post-data "`svnlook changed --revision $REV $REPOS`" \
  --output-document "-" \
  --timeout=2 \
 http://localhost:8080/subversion/${UUID}/notifyCommit?rev=$REV
```

Listing 1

mittelt werden, wenn für Jenkins eine Authentifizierung notwendig ist. Wie das mit einem SVN-Hook gemacht wird, ist in Listing 1 zu sehen. Der Code muss in die *POST-COMMIT*-Datei eingetragen werden.

Ein komplexeres Beispiel

Das bisherige Beispiel war ein einfaches Perl-Modul, das für sich allein getestet und erstellt werden kann. Wie sieht es aber aus, wenn mehrere Module voneinander abhängig sind? Auch hier muss man sich wieder überlegen, wie der Integrationsprozess aussieht.

In diesem Artikel haben wir ja schon zwei Perl-Projekte gezeigt. Jetzt gehen wir davon aus, dass das zweite Projekt erst dann gebaut werden kann, wenn das erste Projekt gebaut wurde. Schon bei der Konfiguration des Jobs kann man angeben, dass der Job erst abgearbeitet wird, wenn andere Projekte gebaut wurden (Abbildung 8).

Für die Planung, welche Projekte in welcher Reihenfolge abgearbeitet werden, lohnt es sich, eine kleine Skizze wie in Abbildung 9 anzufertigen. Dann übersieht man keine Abhängigkeiten.

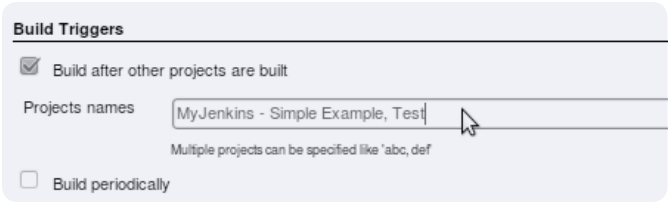


Abbildung 8: Ein Projekt nach dem anderen.

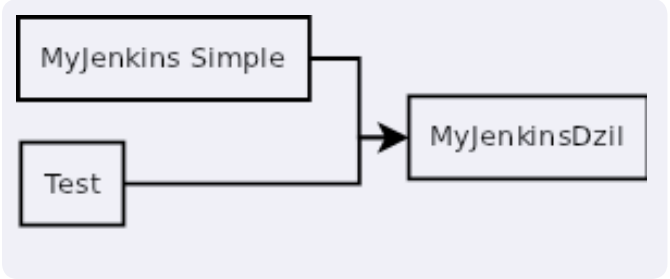


Abbildung 9: Skizze von Abhängigkeiten von Projekten

Noch mehr Infos für Perl-Projekte

Bis jetzt wurden nur ganz einfache Tests für die Perl-Projekte ausgeführt. Aber sind die Tests wirklich ausreichend? Vor allem ist die Frage, ob die Tests wirklich den gesamten Code testen, interessant. Diese Frage kann man mit `Devel::Cover` beantworten. Damit bekommt man eine sehr ausführliche Übersicht, welche Dateien, welche Subroutinen, Zeilen oder `if`-Bedingungen getestet wurden.

Solche Informationen sind der Qualität von Software sehr zuträglich. Denn Entwicklern macht es häufig Spaß, alles mögliche zu versuchen, um an die 100%-Marke zu kommen. Man darf aber nicht glauben, dass eine 100%ige Code-Abdeckung durch Tests gleichbedeutend mit Bug-Freiheit ist. Es kommt natürlich auch sehr auf die Qualität der Tests an, dass Grenzfälle bei den Testdaten berücksichtigt sind usw.

Dazu muss beim Testen einfach ein `HARNESS_PERL_SWITCHES=-MDevel::Cover` übergeben werden:

```
HARNESS_PERL_SWITCHES=
  -MDevel::Cover prove -Ilib -v ...
```

Damit liegen jetzt die Informationen im *workspace* bereit. Um sie dann in Jenkins sichtbar zu machen, müssen sie publiziert werden. Dafür benötigen wir das *HTML Publisher Plugin*, das wir genau wie das Git-Plugin über den Pluginmanager von Jenkins installieren. Leider hat das Plugin in der aktuellen Version 0.6 das Problem, dass man keine Umgebungsvariablen (z.B. `BUILD_NUMBER`) in den Pfaden verwenden kann. Aus diesem Grund muss man schon in den Build-Schritten die Ergebnisse umkopieren (Abbildung 10).

Nach der Installation muss noch eingestellt werden, welche HTML-Dateien veröffentlicht werden sollen (Abbildung 11).

Abbildung 10: Kopieren der Daten in einen festen Pfad

Abbildung 11: Die Ergebnisse von Devel::Cover sollen veröffentlicht werden

Jetzt ist alles eingestellt und die Daten werden veröffentlicht (Abbildung 12).

Von außen auf Jenkins zugreifen

Jenkins bietet auch eine Remote-API, mit der man von außen auf Ergebnisse etc. zugreifen kann. Dies kann man zum Beispiel nutzen, wenn man sein Team motivieren möchte, besseren Code zu schreiben. Dann kann man eine kleine Ampel in den Gang oder ins Büro hängen, die den aktuellen Build-Status anzeigt. Diese Anwendung dient hier als kleines Beispiel dafür, wie man die API von Jenkins nutzt.

```
#!/usr/bin/perl

use strict;
use warnings;
use JSON::PP;
use LWP::Simple;

my $base_url =
   'http://localhost:8080/job/MyJenkinsDzil/
     api/json';
my $json     = get $base_url;

my $perl = decode_json( $json );

print "current state: ",
   $perl->{color},"\n";
```

Wenn man nicht weiß, was man mit der API machen kann, lohnt es sich, sich durch die Weboberfläche zu klicken und immer mal ein `/api/json` anzuhängen.

Aber es gibt nicht nur die Möglichkeit, die JSON-API anzusprechen, sondern auch die Daten im XML-Format zu bekommen.

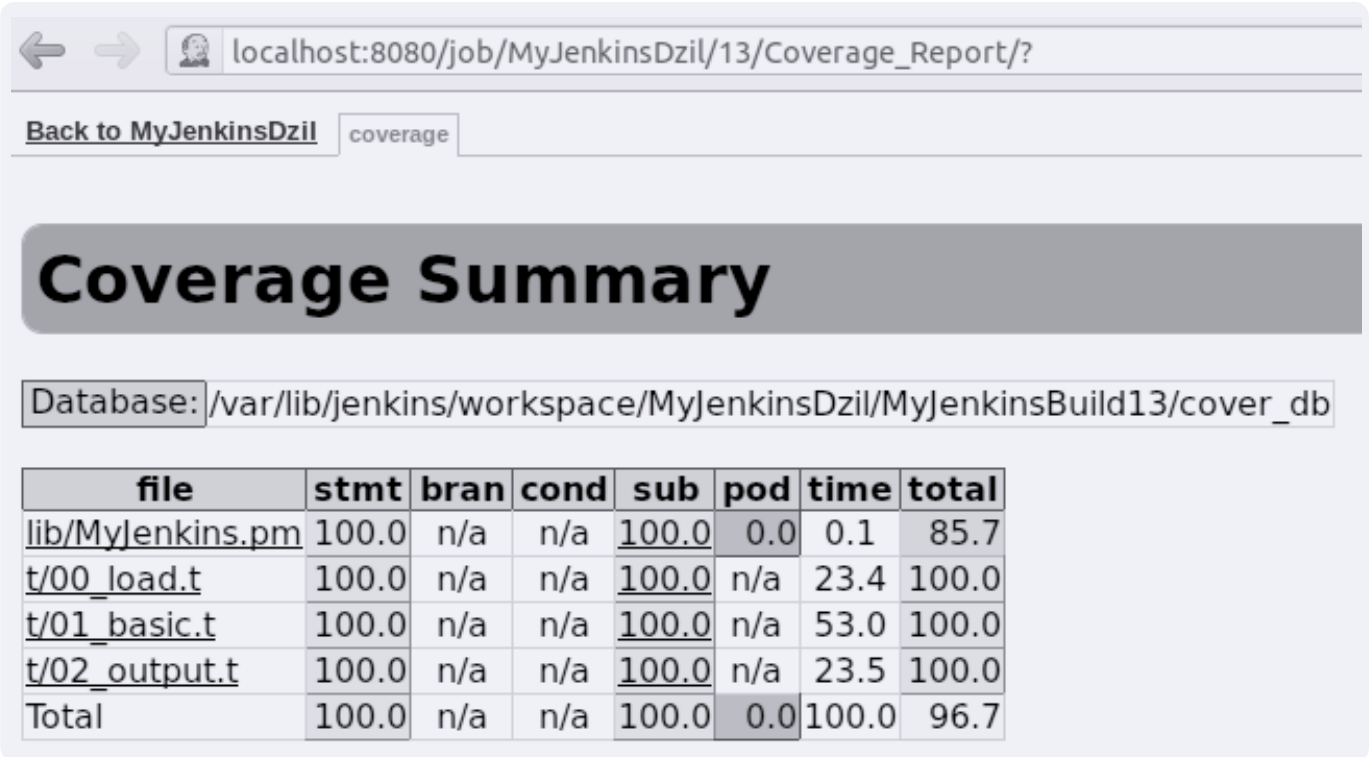


Abbildung 12: Die Testabdeckung im HTML-Format

Fazit

Mit Continuous Integration kann man den Entwicklungsprozess von Softwareprojekten sinnvoll unterstützen. Mit Jenkins steht ein solider CI-Server zur Verfügung, der auch mit Perl-Projekten gut umgehen kann. Nicht umsonst setzen die Perl 5 Porters ebenfalls Jenkins dafür ein, den Perl Kern regelmäßig zu testen. Damit fällt sehr schnell nach einem Commit auf, ob ein Fehler eingebaut wurde oder nicht.

Jenkins kann noch viel detaillierter konfiguriert werden und es bietet weitreichende Möglichkeiten für Plugins. Das würde aber den Rahmen dieses Artikels sprengen. Hier sollte ein Einstieg gegeben werden, der auch bei einfachen Projekten schon ausreichend ist. Leider sieht es bisher mit Literatur eher mager aus, aber ich kann das Buch *Continuous Integration mit Hudson* von Simon Wiest empfehlen.

Links

http://blogs.perl.org/users/confuseacat/2011/09/perl-testing-with-jenkinshudson-avoiding-some-pitfalls.html
http://logiclab.jira.com/wiki/display/OPEN/Continuous+Integration
http://my.opera.com/cstrep/blog/2011/03/08/continuous-integration-of-perl-based-projects-in-hudson-jenkins
http://www.slideshare.net/jonasbn/perl-and-jenkins-for-osd2011
https://wiki.jenkins-ci.org/display/JENKINS/Plugins
http://www.howtoforge.com/continuous-deployment-with-jenkins-and-rex
http://isaac.inkrunway.com/?p=100
http://en.wikipedia.org/wiki/Continuous_integration
https://gist.github.com/812398
http://www.squidoo.com/hudson---my-favourite-developer-tool-in-2011
https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin
http://amokti.me/2011/10/11/automatic-opt-in-branch-building-with-jenkins-and-git-2/
http://blog.crimeminister.org/post/4177778711/triggering-jenkins-builds-from-a-git-hook
http://www.nomachetejuggling.com/2011/07/31/ubuntu-tomcat-jenkins-git-ssh-togethe/
http://jenkins-ci.org/
http://labs.opsview.com/2011/01/grails-hudson-jenkins-part-5-monitoring-build-status/
http://blog.crimeminister.org/post/4177778711/triggering-jenkins-builds-from-a-git-hook
http://kohsuke.org/2011/12/01/polling-must-die-triggering-jenkins-builds-from-a-git-hook/

Thomas Fahle

Perl in the Cloud - OpenShift Express by Red Hat

OpenShift by Red Hat soll die Entwicklung von Open-Source-Anwendungen für die Cloud vereinfachen. Diese *Platform as a Service* stellt eine Infrastruktur für verschiedene Programmiersprachen, Web-Frameworks und Open-Source-Anwendungen zur Verfügung.

Red Hat unterscheidet zwischen den Produktvarianten Express, Flex und Power. Die kostenlose Express Variante, für die nur eine Registrierung erforderlich ist, erlaubt u.a. die Verwendung der dynamischen Programmiersprache Perl in Version 5.10.1, der Datenbanken MySQL in Version 5.1, SQLite in Version 3, MongoDB in Version 2 und (ganz wichtig) die Installation von CPAN-Modulen. Die bekannten und beliebten Perl-Web-Frameworks Dancer, Mojolicious und Catalyst können verwendet werden.

Schritt für Schritt

Dieser Beitrag geht zuerst Schritt für Schritt durch die Installation, Initialisierung und Konfiguration der Clientprogramme von OpenShift Express. Danach wird ein betont einfaches, aber nützliches Perl-Programm erstellt und gezeigt, wie dieses in die Cloud ausgeliefert wird (Deployment). Im letzten Abschnitt werden noch ein paar kleine Tricks gezeigt, die den Umgang mit einer App vereinfachen.

Registrierung

Zur Verwendung von OpenShift Express ist eine Registrierung mit einer gültigen E-Mail-Adresse erforderlich. Alternativ kann auch ein bestehender Account des Red Hat Network (RHN) verwendet werden.

Installation der Client-Tools

OpenShift Express wird über eine Sammlung von Kommandozeilen-Tools verwaltet. Die Client-Tools sind in Ruby geschrieben und laufen auf Mac OSX, Linux und Windows (Cygwin). Zum Deployment der Apps wird Git verwendet. Die Kommunikation zwischen Client und OpenShift ist per SSH verschlüsselt.

Installation unter Red Hat/CentOS/Fedora

Für die Client-Tools bietet RedHat ein eigenes YUM-Repository an, das wie folgt installiert wird.

```
# wget https://openshift.redhat.com/app/
                              repo/openshift.repo
# mv openshift.repo /etc/yum.repos.d
```

Damit alle Paketabhänigkeiten aufgelöst werden, musste ich auch die EPEL- und RPMForge-Repositories hinzufügen. Nun können die Client-Tools installiert werden:

```
# yum install rhc
```

Die Clientprogramme befinden sich nun im Ordner `/usr/bin/` und sind damit direkt aufrufbar, da sie im Pfad enthalten sind.

Installation unter Ubuntu/Debian/SUSE

Für Ubuntu/Debian/SUSE werden Ruby in Version 1.8 einschließlich Entwicklerpaketen, Rubygems, ein passendes OpenSSL und Git benötigt. Nach der Installation der Pakete können die Client-Tools als `rubygems` installiert werden.

Beispielinstallation unter Ubuntu 10.04 LTS

```
sudo apt-get install ruby ruby-dev
sudo apt-get install libopenssl-ruby
sudo apt-get install rubygems
sudo apt-get install git-core

sudo gem install rhc
```

Die Clientprogramme befinden sich nun im Ordner */var/lib/gems/1.8/bin/* und sind nur über die Angabe des kompletten Pfades erreichbar.

Abhilfe schafft hier eine Änderung der Umgebungsvariablen PATH:

```
export PATH="/var/lib/gems/1.8/bin:$PATH"
```

Diese Änderung lässt sich auch in *~/.bashrc* dauerhaft ablegen.

Initialisierung - Namensraum festlegen

Alle Apps eines Users werden in einen eigenen Namensraum (Domain) installiert. Apps sind dann nach dem Schema http://$app-$domain.rhcloud.com öffentlich erreichbar. Dazu gleich mehr.

Das Kommando *rhc-create-domain* erzeugt einen neuen Namensraum, die Konfigurationsdatei *express.conf* und den SSH-Schlüssel *libra_id_rsa* zur Git-Authentifizierung. Über die Option *-n* wird der Namensraum festgelegt, die Option `-l` nimmt den OpenShift-Benutzernamen entgegen.

Weitere Optionen können über den Schalter *-h* angezeigt oder der manpage zu *rhc-create-domain* entnommen werden.

```
$ rhc-create-domain -n yourdomain \
    -l user@example.com
Password: <user password>

Generating Openshift Express ssh key to
  /home/UserName/.ssh/libra_id_rsa
Generating public/private RSA key pair.
Created directory '/home/UserName/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
  /home/UserName/.ssh/libra_id_rsa.
Your public key has been saved in
 /home/UserName/.ssh/libra_id_rsa.pub.
.
.
Contacting https://openshift.redhat.com
Adding rhlogin to
  /home/UserName/.openshift/express.conf
Creation successful

You may now create an application.
  Please make note of your local config file
  in /home/UserName/.openshift/express.conf
  which has been created and populated for
  you.
```

Jetzt noch Git (minimal) konfigurieren:

```
$ git config
    --global user.name "Your Name"
$ git config
    --global user.email you@example.com
```

Dann kann endlich die erste App erstellt werden.

App-Gerüst erzeugen

Das Kommando *rhc-create-app* erzeugt das Gerüst der neuen App. Über die Option *-a* wird der Name der Applikation angegeben. Die Option *-t* legt den Typ der Applikation, hier *perl-5.10*, fest. Typen werden in der Dokumentation auch gerne als `cartridge` bezeichnet.

```
$ rhc-create-app -a X1 -t perl-5.10
Password:

Attempting to create remote
application space: X1
Now your new domain name is being
propagated worldwide
(this might take a minute)...
Pulling new repo down
...
Confirming application 'X1' is available
 Attempt # 1

Success!  Your application 'X1' is now
 published here:

     http://X1-perlhowto.rhcloud.com/
...

To make changes to 'X1', commit to X1/.
Successfully created application: X1
```

Die neu erstellte Applikation mit dem Namen *X1* innerhalb des von mir gewählten Namensraumes *perlhowto* ist sofort unter der URL *http://x1-perlhowto.rhcloud.com/* erreichbar. Alle Applikationen sind auch über SSL (https) erreichbar.

Wer lieber seine eigene Domain verwenden möchte und über einen eigenen Nameserver verfügt, kann einen DNS-Alias (CNAME) einrichten. Eine Anleitung befindet sich in den OpenShift Express FAQs.

Orientierung im Gelände

Die neu erstellte Applikation befindet sich im Verzeichnis $app, hier X1.

```
$ tree X1
X1
|-- deplist.txt
|-- libs
|-- misc
|-- perl
|   |-- health_check.pl
|   `-- index.pl
`-- README
```

Das Verzeichnis *perl* ist die DocumentRoot der Webapp. Alle hier abgelegten Dateien sind öffentlich. Die Datei *index.pl* dient als DirectoryIndex.

Die Datei *deplist.txt* nimmt eine Liste der zu installierenden CPAN-Module auf, pro Zeile ein Modul ohne Versionsnummer.

Beispiel:

```
YAML
Dancer
Plack::Handler::Apache2
```

Die hier angeführten Module werden auf dem Cloud-Server mittels *cpanm* installiert.

Das Verzeichnis *misc* ist nicht öffentlich und kann für eigene Zwecke genutzt werden.

Das ebenfalls nicht öffentliche Verzeichnis *libs* dient als Speicherort für eigene Perl-Module.

Der Sinn und Zweck der Datei *health_check.pl* ist mir nicht ganz klar geworden, das Programm kann aber zum Monitoring eingesetzt werden.

Weiterhin gibt es noch ein verstecktes Verzeichnis *.openshift* zur Steuerung des Build-Prozesses. Dazu später mehr.

Das erste Programm: Umgebungsvariablen

Einige Konfigurationseinstellungen, z.B. für Datenbanken, sind als Umgebungsvariablen abgelegt.

Daher erstellen wir als erstes einfaches Beispiel keine Hallo-Welt-App, sondern eine nützliche App, welche die Umgebungsvariablen anzeigt.

Das Programm wird unter dem Namen *printenv.pl* im Ordner *perl* abgelegt.

```
#!/usr/bin/perl
use strict;
use warnings;

# printenv -- demo CGI program which
# just prints its environment
print "Content-type: text/plain\n\n";

foreach my $key ( sort( keys(%ENV) ) ) {
   my $val = $ENV{$key};
   $val =~ s|\n|\\n|g;
   $val =~ s|"|\\"|g;
   print qq~$key = $val\n~;
}
exit();
```

Zur Veröffentlichung (Deployment) der App verwendet OpenShift Git. Sobald die Datei hinzugefügt und commited wurde, kann diese per *git push* in die Cloud ausgeliefert werden.

```
$ git add printenv.pl
$ git commit -m 'Umgebungsvariablen App'
$ git push

...
remote: Stopping application...
remote: Waiting for stop to finish
remote: Done
...
```

Das Programm ist nun unter der URL *http://x1-perlhowto.rhcloud.com/printenv.pl* erreichbar.

Hinweis: Alle Daten innerhalb des Git-Repositories werden dabei auf dem OpenShift Express Server zunächst gelöscht und dann neu eingespielt.

Hinweis: Da Umgebungsvariablen auch sensible Daten enthalten können, sollte dieses Programm nicht auf dem Cloud-Server verbleiben.

Zum Bau und zur Auslieferung der Applikation werden die Programme im Ordner *.openshift/action_hooks/* ausgeführt. Um sich die Umgebungsvariablen anzeigen zu lassen, genügt es in die Datei *build* die Anweisung `export` einzufügen. Dann werden die Umgebungsvariablen bei jedem push angezeigt.

```
$ cat .openshift/action_hooks/build

#!/bin/bash
# This is a simple build script and will
# be executed on your CI system if
# available.  Otherwise it will execute
# while your application is stopped
# before the deploy step.  This script
# gets executed directly, so it
# could be python, php, ruby, etc.
export
```

Eigene Umgebungsvariablen können derzeit nicht gesetzt werden.

Logbuch-Dateien

Das Kommando *rhc-tail-files* ermöglicht den Zugriff auf die Logbuch-Dateien auf dem Cloud-Server.

```
$ rhc-tail-files -a X1
Password:

Attempting to tail files: X1/logs/*
Use ctl + c to stop

==> X1/logs/error_log-... <==
[Date/Time] [notice] ...

==> X1/logs/access_log-... <==
xx.xxx.xxx.IP - - [Date/Time] \
 "GET /printenv.pl HTTP/1.0" 200 2323 "-" "
```

Der Zugriff auf die *error_log* Dateien erleichert das Debuggen erheblich.

Snapshots

Das Kommando *rhc-snapshot* erstellt einen Snapshot der Applikation und liefert diesen als gezippte *tar*-Datei zurück:

```
$ rhc-snapshot  -a X1
Password:

Pulling down a snapshot to X1.tar.gz

Stopping application...
Waiting for stop to finish
Done
Creating and sending tar.gz
Starting application...
Done
```

Wenn man die Datei *X1.tar.gz* auspackt, sieht man **alle** Verzeichnisse und Dateien der Applikation:

```
$ tree
.
|-- git
`-- X1
    |-- ci
    |-- conf
    |   `-- magic -> /etc/httpd/conf/magic
    |-- conf.d
    |-- data
    |-- logs
    |-- modules -> /usr/lib64/httpd/modules
    |-- perl5lib
    |-- repo -> runtime/repo
    |-- run
    |-- runtime
    |   `-- repo
    |       |-- deplist.txt
    |       |-- libs
    |       |-- misc
    |       |-- perl
    |       |   |-- health_check.pl
    |       |   |-- index.pl
    |       |   `-- printenv.pl
    |       `-- README
    `-- tmp
```

Die Ausgabe der Verzeichnisauflistung habe ich für diesen Artikel deutlich gekürzt.

Persistent Storage

Wie oben bereits erwähnt werden bei der Auslieferung per `git push` alle Dateien, die sich innerhalb des Git-Repositories befinden, auf dem Cloud-Server gelöscht und neu eingespielt. Persistente Daten, z.B. SQLite Dateien, müssen daher außerhalb des Git-Repositories auf dem Server aufbewahrt werden. Dazu stellt OpenShift Express den Ordner *data* zur Verfügung.

Der Pfad zum Ordner *data* kann aus der Umgebungsvariablen *OPENSHIFT_DATA_DIR* ermittelt werden.

```
#!/usr/bin/perl
use strict;
use warnings;

print "Content-type: text/plain\n\n";

my $data_dir = $ENV{OPENSHIFT_DATA_DIR};
my $file = 'test.txt';

open (OUT, ">" , "$data_dir/$file" )
      or die $!;

for ( 1 .. 10 ) {
       print OUT "$_\n";
}
close(OUT) or die $!;
print "Okay. Datei $file angelegt.";
exit();
```

Die im Ordner *data* gespeicherten Daten lassen sich per *rhc-snapshot* vom Cloud-Server holen.

```
$ tree
.
|-- data
|   `-- test.txt
```

Weitere Cartridges

Applikationen lassen sich mit dem Kommando *rhc-ctl-app* anpassen. Der Parameter *-L* listet alle verfügbaren Ergänzungen auf:

```
$ rhc-ctl-app -a X1 -L

List of supported embedded cartridges:

Obtaining list of cartridges ...
jenkins-client-1.4, mongodb-2.0,
rockmongo-1.1, metrics-0.1, mysql-5.1,
phpmyadmin-3.4
```

Cartridges werden in eine bestehende App eingebettet (*-e*).

```
$rhc-ctl-app -a X1 -e add-mysql-5.1
Password:

RESULT:

Mysql 5.1 database added.
Please make note of these credentials:

  Root User: admin
  Root Password: .......
  Database Name: X1

Connection URL: mysql://127.1.21.1:3306/

You can manage your new Mysql database by
also embedding phpmyadmin-3.4.
```

Dieses Beispiel erzeugt eine MySQL-Datenbank ohne Tabellen, diese müssen durch ein eigenes *create tables* Programm erzeugt werden.

END {}

OpenShift Express ist eigentlich einfach zu benutzen. Dieser Artikel enthält sehr wenig Perl-Code, aber den kann der Leser ja selber schreiben.

Herbert Breunung

Rezension - Perl komplett

Johan Vromans
Perl Pocket Reference
5. Auflage Juli 2011
102 Seiten, englisch
O'Reilly
ISBN 978-1-449-30370-9 $12,99
E-Book: ISBN 978-1-4493-0813-1 $7,99

Jürgen Plate
Der Perl-Programmierer
Perl lernen – Professionell anwenden – Lösungen nutzen
1. Auflage Juli 2010
1.232 S., Flexcover
Hanser Verlag
ISBN 978-3-446-41688-8
€59,90

Übersicht

Die Frühjahrsausgabe dieser gemütlichen Bücherecke geht der Frage nach, was ein vollständiges Perlbuch ausmacht. Doch zuvor ein kleiner Spaziergang auf den Markt. Die neuen Webstandards sind natürlich hier ein größeres Thema. O'Reilly [1] hat seine CSS-Taschenreferenz aktualisiert und das 600-seitige Buch *Head First HTML5 Programming* herausgegeben, vorerst nur auf englisch.

Galileo behandelt beides mit der neunstündigen Lehr-DVD mit dem schlichten Namen *HTML5 und CSS3* von Peter Kröner, hat aber auch "Fortgeschrittene CSS-Techniken" von Chao und Rudel mit Beispiele- und Video-DVD. Zum verwandten Web-Thema `NoSQL` brachte Hanser [2] ein gleichnamiges deutsches Buch auf den Weg und O'Reilly eines über Hadoops NoSQL-Datenbank *HBase* (englisch).

Wer auf der Ebene darunter unterwegs ist, muss Ahnung von den Transportprotokollen haben. Der mitp-Verlag brachte mit der *TCP/IP - Studienausgabe* ein detailreiches und mit 24,95€ recht günstiges Werk heraus. Im gleichen Haus erschien die Neuauflage des einer Taschenreferenz ähnlichen Buches *vim 7.3 GE-PACKT*.

Linux ist immer ein Thema. Das wusste auch der Verlag Addison-Wesley, der *Linux-Firewalls* von Ralf Spenneberg und *Linux 2012* drucken lässt. Galileo stellt dazu *Linux: Das umfassende Handbuch* und einen Schmöker (1024 Seiten+DVD) über *Apache 2.4* von Sascha Kersken vor. Zu guter Letzt hat O'Reilly nicht nur – wie letztens erwähnt – die aktualisierte *Einführung in Perl* übersetzt, sondern auch eine Initiative in Sachen Design und Softwarequalität gestartet.

Von den drei Büchern *Designed for Use* (Wie sorge ich dafür, dass die Oberfläche keine Nutzer verschreckt?), *Codemetrics* (Woran kann man fehlerhaften und problemanfälligen Code erkennen?) und *The Art of Readable Code* hatte ich mir letzteres angesehen. Es ist ein absolutes Einsteigerbuch, das man nicht mit den Klassikern wie dem *Clean Code* (besprochen in $foo 2/2011) vergleichen kann. Denn es geht weniger auf größere Zusammenhänge ein, sondern beschränkt sich auf die Vermeidung der wesentlichen, leichter vermeidbaren Fallen. Diese werden aber knapp und unterhaltsam vorgestellt und sind für Aspiranten immer noch genügend Stoff, der ein bis zwei Jahre zur Verinnerlichung benötigt. Dank der Kürze und Anschaulichkeit, die mit einfacher Sprache, deutlichen Beispielen und eingeflochtenen Comicstreifen erreicht wurde, ist es durchaus ein Buch, das noch fehlte und Anfängern

dringend anzutragen ist. Trotzdem sollten Perl-Programmierer eher zu *Perl Best Practices* greifen, weil es viele der Ratschläge auch und noch einige mehr beinhaltet und speziell auf die vielen Eigenheiten von Perl eingeht. In diesem Werk sind die Beispiele in Python, C++, Java und Javascript geschrieben.

Die Rezension der Perl-Bibel schlechthin wird verschoben, da dieses Heft wahrscheinlich noch vor der fünften Edition von *Programming Perl* erscheint.

Taschenreferenz

Kaum zu glauben, aber von der Seite gesehen, ist es vielleicht das älteste Perlbuch [3] überhaupt. Bereits um die Zeit der Version 2.0 (Ende 1988, das Kamel kam drei Jahre später) erschien im Netz der dreispaltige Spickzettel, aus dem sich die kompakte Übersicht des Sprachkerns entwickelte, die endlich 1996 in Buchform gebracht wurde. Sie wurde alle zwei Jahre aktualisiert, was allerdings vor neun Jahren einschlief. Das lag jedoch nicht daran, dass der alte Hase Johan sein Interesse an Perl verlor, sondern dass er seine Aufgabe sehr ernst nimmt. Während der weitreichenden Recherche zu dieser fünften Auflage gab es lange Diskussionen in der Mailingliste der Perlentwickler `p5p`, die zu Verbesserungen in den *perldoc*-Seiten führten. Denn Perl 5.10 und folgende brachten viel Neues, was noch nicht überall in der offiziellen Dokumentation ankam. Diese pflegt überdies manchmal einen Stil, die heute nur noch in kleineren Programmen vertretbar ist. Zum Glück ist vor allem brian d foy seit einiger Zeit damit beschäftigt, diese teilweise sehr kontroversen Probleme nach und nach mit 5.14, 5.14.1 und weiteren Versionen zu beheben.

Die Taschenreferenz ist ein wenig wortreicher und wesentlich umfangreicher verfasst als die `perldoc perlcheat`. Sie ist ein Stichwortgeber, den man aufsucht um die Schreibweise einer Spezialvariable oder magischen Konstante nachzuschlagen. Manchmal erinnert man sich nicht, ob der letzte Zugriff auf eine Datei das siebente oder achte Arrayelement von `stat` birgt (es ist Nummer acht). Dann ist es praktisch, dieses handtellergroße Büchlein neben dem Rechner liegen zu haben. Auch der Autor dieser Rezension lernte auf diese Weise wesentliche Feinheiten der Syntax. Neu hinzu kamen Tabellen zu den Funktionen des Smartmatch-Operators und der IO-Layer, die neuen gierigen Quantoren, die experimentellen Steuersymbole fürs *Backtracking* usw. Die Linkliste musste sich sehr verändern und eine ganze Reihe Fehler oder fehlender Details, die sich all die Jahre und Auflagen hielten, wurde ausgebessert. Die wenig sagende Liste mit den Kernmodulen wurde gestrichen, wodurch die Seitenzahl sogar auf 93 nummerierte abnahm. Es ist die dünnste von allen Taschenreferenzen und ein paar Kleinigkeiten könnte man vermissen, wie die Rückgabewerte von `ref`. Mir fehlte leider auch die witzige Einleitung von Larry Wall, die nur in der Version für Perl 5.6 aus dem Jahr 2000 zu finden ist. Ebenfalls entschied sich Vromans strikt dagegen, irgendwelche Module zu beschreiben, so verbreitet und nützlich sie auch sein mögen (manche Kernmodule werden nur kurz angerissen). Er wünscht, dass weitere Referenzen zu Modulen oder Themen geschrieben werden. Ein passender Platz diese zusammenzutragen wäre das kürzlich gegründete *http://perl-tutorial.org/*. Die englische Ausgabe der Taschenreferenz erschien bereits im Juli und die deutsche wird wohl ab März 2012 erhältlich sein.

Der Perl-Programmierer

Beinahe unbemerkt wurde im Sommer 2010 ein Perl-Buch in die Regale geschoben, dessen Konzept großes Potential birgt. Es will alles mitbringen, um aus einem Einsteiger einen kompletten Perl-Programmierer zu machen (daher der Titel). Jemanden, der die klassischen Skripte für Netzwerk, Mail und CGI im Halbschlaf schreiben kann. In drei Teilen lehrt es auf vielen Seiten die Grundlagen der Sprache, das Schreiben von Modulen und Objekten und die Anwendung der wichtigsten Module für den praktischen Einsatz. Dazu bietet es noch etwas Hintergrundinformationen zur Programmierung allgemein sowie zu den verwendeten Protokollen, die Professor Plate an der Universität München lehrt und noch genauer in anderen Büchern beschrieben hat.

Die reichlichen Beispiele lassen sich kopieren, erweitern und sofort nutzen. Es ist ein Nachschlagewerk für die praktische Arbeit. Darüber hinaus gibt es auf der Netzseite des Autors [4] noch einige Zusatzkapitel. Dabei ist auch eine kleine Taschenreferenz, die alle Funktionen und Kernmodule zusammenfasst. Die Liebe zu den Details zeigt sich bereits in den Zitaten, mit denen die Kapitel beginnen und die eines Damian Conway würdig wären. Eine Kostprobe:

Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vaccuum tubes and perhaps weigh 1.5 tons.

Popular Mechanics, März 1949

Überhaupt wäre Jürgen Plate der richtige Mann dafür, einen unterhaltsamen Wälzer zu schreiben, der allen O'Reilly-Titeln trotzen könnte. Zu Recht schreibt der Hanser-Verlag über ihn: "Seit vielen, vielen Jahre ist er begeisterter Perl-User". Er hat ein Gefühl für die Perl-Kultur und gibt nicht viel auf oberflächliche Trends. Sein sachlicher, aber beschwingter Erzählton sucht keine Pointen, sondern kommentiert nur leicht ironisch, was die Wirklichkeit an Widersprüchen bietet. Das macht ihn teilweise angenehmer zu lesen als manchen der großen Namen. Dabei ist auch seine Lehrerfahrung spürbar. Kaum eine andere Perl-Fibel nimmt sich den Platz, die benutzten Technologien zu erklären. Der Anhang zeigt darüber hinaus, dass Plate die Literatur gut kennt.

Und es muss wirklich nicht immer ein Einband mit einem Tier sein, selbst wenn das Buch in diesem Haus unter *Andere Programmiersprachen* im Katalog eingeordnet ist. Hanser hat zum Beispiel mit Kernighans und Ritchies *Programmieren in C* selbst Klassiker im Angebot und die Reihe *Der ...-Programmierer* ist eine ernsthaft geführte Serie, deren C++-Ausgabe erst dieses Jahr mit der Erneuerung des Standards eine Neuauflage erfuhr.

Trotzdem ist dieses Werk nur eingeschränkt empfehlenswert. Zum Beispiel steht auf dem Umschlag, es sei *ein Praxisbuch für alle Ansprüche – mehr brauchen Einsteiger und Profis nicht*. "Die Perl-Programmierer" benutzen heute `Dist::Zilla`, `Catalyst`, `Mojolicious` oder `Dancer`, `Plack`, `Wx` oder `GTK`, `POE`, `AnyEvent` und vieles mehr. Nichts davon wird im Text erwähnt. Das ist allerdings nicht unbedingt ein Versäumnis, denn die inhaltliche Beschränkung entspricht grob einem verbreiteten Nutzerprofil und macht sehr viel Sinn. Nur hätte man es ehrlicher ausweisen sollen als "umfassende Anleitung für werdende sehr gute Programmierer, welche die meisten traditionellen Aufgaben mit Sicherheit lösen können".

Für mein Empfinden unterscheidet einen "Programmierer" von einem "Skripte-Schreiber", dass er auch mit größeren Strukturen und Projekten umgehen kann. Themen wie Dokumentation, Objekte, das Testen und der Debugger wurden dazu behandelt, Ausnahmebehandlung, Profiling und Benchmarking allerdings nicht. Dafür wurden angenehm viele Perl-Spezifika wie `AUTOLOAD`, *Taint*, *Typeglobs* und höhere *Regex*-Funktionen vorgestellt, seltsamerweise nicht `Tie`. Insgesamt wurde großer Wert darauf gelegt, jeden scheinbar unbedeutenden Zwischenschritt (Wie schreibe ich eine *README*?) zu erläutern.

Wirklich schmerzlich war jedoch die Empfehlung von `Switch`, was nach langen Ankündigungen und Warnungen vor den Nebenwirkungen mit Erscheinen des Buches aus dem Perl-Kern entfernt wurde.

`given`/`when` wurde nur im Nebensatz erwähnt, seine vielen praktischen Möglichkeiten blieben verschwiegen. Das gilt für alles, was mit Version 5.10 kam (siehe $foo-Ausgaben 1 und 2 2007). Da "Der Perl-Programmierer" zweieinhalb Jahre später veröffentlicht wurde, ist das kaum entschuldbar. Auch `Moose`, `Try::Tiny`, `autodie`, `Perl::Critic` und vieles mehr waren 2010 kein Geheimtipp mehr. In vielen weiteren Details merkt man, dass Herr Plate nicht am Puls der Entwicklung lebt. (Selbst bei einer Entscheidung für `Tk` gehört unbedingt `Tkx` dazu.) Täte er das, wäre es mit ein wenig optischer Politur ein rundum gutes, vielleicht sogar maßgebliches Buch für Anfänger bis weit Fortgeschrittene geworden. Leider ist eine Neuauflage derzeit nicht geplant.

Links

[1] http://www.oreilly.de/catalog/
[2] http://www.hanser.de/nb.asp?area=Computer
[3] http://www.vromans.org/johan/perlref-history.html
[4] http://www.netzmafia.de/skripten/buecher/index.html

Jan Gehring

Konfigurationsmanagement und Software-Deployment mit Rex

Rex entstand vor ca. einem Jahr aus dem Bedürfnis heraus, ein Werkzeug an der Hand zu haben mit dem man auf mehreren Servern der Reihe nach Befehle ausführen kann. Daher auch der Name Rex, der für Remote Execution steht. Konkret ging es darum ein altes Deployment Script zu ersetzen. Die erste Version konnte auch nur genau das: einloggen auf mehreren Remote-Systemen per SSH, Befehle ausführen und die Ausgabe anzeigen. Mit der Zeit sind immer mehr Befehle dazu gekommen und es kommen auch immer neue Ideen von Benutzern dazu.

Rex kann mittlerweile das komplette Deployment und die Konfiguration von Servern erledigen. Außerdem gibt es ein spezielles Modul zum Ausrollen von Webapplikationen. Rex ist für viele Distributionen und über CPAN verfügbar. Die Kompatibilitätsmatrix kann unter [1] eingesehen werden. Rex steht unter der Lizenz GPLv3.

Die Steuerung von Rex wird über sogenannte Tasks vorgenommen. Um einen Task abzuarbeiten verbindet sich Rex mittels SSH auf das Zielsystem und führt die gegebenen Befehle aus. Rex ist eine Art `make`. Der Unterschied ist nur, dass die Sprache der Steuerdatei hier Perl ist.

Installation

Rex Pakete stehen für die 64 Bit-Systeme Debian (Lenny und Squeeze), Fedora 15, OpenSuSE 11.4, CentOS (5 und 6), Ubuntu LTS (10.04 und 11.04) und Mageia bereit. Wie man die Paketmanager-Quellen einbindet kann unter [2] nachgelesen werden. Außerdem steht ein PPD-Paket für Windows ActiveState Perl zur Verfügung.

Nach dem Einbinden der Paketquellen wird Rex über die normalen Bordmittel der Distribution installiert (apt-get, yum, ...). Die Installation aus den Quellen ist auch nicht weiter schwierig, wie man hier anhand von cpanmin.us sehen kann. Diese Befehle müssen als *root* ausgeführt werden.

```
curl -L cpanmin.us | perl - Rex
```

Es wird noch das Deployment-Modul von Rex benötigt.

```
curl -L cpanmin.us |
              perl - Rex::Apache::Deploy
```

Auf einem Debian/Ubuntu System müssen folgende Pakete installiert sein, damit der Installationsprozess alle Abhängigkeiten installieren kann.

- build-essential
- libexpat1-dev
- libssh2-1-dev
- zlib1g-dev
- libssl-dev

Erste Schritte

Als Erstes erstellen wir einen kleinen Task, der sich auf das Zielsystem verbindet und uns die *uptime* ausgibt. Dazu legen wir eine Datei *Rexfile* an und übernehmen folgende Zeilen.

```
1 user "root";
2 password "geheim";
3
4 task "uptime", "server1", sub {
5    say run "uptime";
6 };
```

Diesen Task können wir jetzt mit folgendem Befehl ausführen:

```
rex uptime
```

In den ersten beiden Zeilen werden die Authentifizierungsoptionen gesetzt: Benutzer (Zeile 1) und Passwort (Zeile 2). Werden kein Benutzer und kein Passwort angegeben, versucht sich Rex mit dem aktuell eingeloggten Benutzer und dem SSH-Key $HOME/.ssh/id_rsa auf das Zielsystem zu verbinden.

Wenn man lieber die Authentifizierung mittels eines SSH-Keys machen möchte, ist das auch kein Problem:

```
1 user "root";
2 password "passphrase des keys";
3 private_key "/pfad/zum/key/id_rsa_home";
4 public_key "/pfad/zum/key/id_rsa_home.pub";
```

Ist der Key ohne Passwortschutz erstellt worden, kann die zweite Zeile weggelassen werden.

Die Authentifizierung mittels SSH-Agent sollte ab Version 0.40 von `Net::SSH2` funktionieren, das konnte ich aber bis jetzt noch nicht testen.

In Zeile 5 wird der Task *uptime* erstellt. Der zweite Übergabeparameter ist der Server, auf dem der Task ausgeführt werden soll. Der letzte Übergabeparameter ist eine anonyme Funktion. Man kann hier so viele Server auflisten wie man möchte:

```
5 task "uptime",
6    "server1", "server2", "server3", sub {
7    say run "/usr/bin/uptime";
8 };
```

Rex beherrscht auch Bereichsangaben beim Servernamen. So ist es einfach eine Gruppe von Servern anzusprechen:

```
5 task "uptime", "server[1..8]", sub {
6    say run "/usr/bin/uptime";
7 };
```

Oder, falls mit Nullen aufgefüllt werden soll:

```
5 task "uptime", "server[01..08]", sub {
6    say run "/usr/bin/uptime";
7 };
```

Ebenso lassen sich Gruppen von Servern definieren, z.B. bei Tasks die auf mehreren Servergruppen laufen sollen:

```
1 group "webserver" =>
2   "middleware[01..10]", "frontend[01..04]";
3 task "prepare-apache", group =>
4    "webserver", sub {
5    install package => "apache2";
6 };
```

Da der Gruppendefinition ein simples Array übergeben wird ist es auch möglich eigene Formate zu erstellen:

```
1 group "webserver", map { "web$_" } (1,2,3);
```

Arbeiten mit Dateien und Verzeichnissen

Rex überschreibt viele interne Perl-Funktionen um ein transparentes Arbeiten zu ermöglichen.

Möchte man eine bestimmte Verzeichnisstruktur anlegen oder Rechte auf Dateien/Verzeichnisse setzen, dann geht das mit den bekannten Perl-Funktionen:

```
1 task "prepare", group => "webserver", sub {
2    mkdir "/var/www/html";
3    chown "www-data", "/var/www/html";
4 };
```

Allerdings erweitert Rex auch verschiedene Funktionen mit zusätzlichen Optionen:

```
1 task "prepare", group => "webserver", sub {
2    mkdir "/var/www/html",
3       owner => "www-data",
4       group => "www-data",
5       mode  => 700;
6
7    chown "www-data", "/usr/lib/cgi-bin",
8       recursive => 1;
9 };
```

Soll eine Konfigurationsdatei auf mehrere Server verteilt werden, steht einem die *file* Funktion zur Verfügung:

```
1 task "configuration",
2    group => "webserver", sub {
3    # remote pfad
4    file "/var/www/conf/db.conf",
5    # lokaler quell pfad
6       source => "files/db.conf",
7       owner  => "root",
8       mode   => 600;
9 };
```

Außerdem besteht die Möglichkeit Templates zu verwenden und Aktionen auszuführen, falls sich die Datei ändert:

```
task "configuration",
  group => "webserver", sub {
  file "/var/www/conf/db.conf",
  content => template("files/db.conf.tpl",
  host => "db01.lan",
  user => "dbuser",
  password => "dbpassword"),
  owner     => "root",
  mode      => 600,
  on_change => sub {
     say "Da hat sich etwas geändert...";
   };
};
```

Das Template kann dann folgendermaßen aussehen:

```
db:
  host: <%= $::host %>
  user: <%= $::user %>
  password: <%= $::password %>
```

Rex stellt eine Reihe von vordefinierten Variablen zur Verfügung wie z.B. die IP-Adressen der Netzwerkkarten. Vordefinierte Variablen sind:

- etho_ip
- etho_mac
- etho_netmask
- etho_broadcast
- architecture
- manufacturer
- hostname
- domain
- operatingsystem
- operatingsystemrelease
- kernel
- kernelname
- kernelversion
- kernelrelease
- memory_cached
- memory_total
- memory_shared
- memory_used
- memory_buffers
- memory_free
- swap_used
- swap_free
- swap_total

Falls vorhanden, gibt es auch eth1_ip, eth2_ip, bondo_ip usw.

Der Deployment-Prozess

In größeren Unternehmen ist es oft so, dass es eine Trennung zwischen dem Betrieb der Server und der Entwicklung gibt. Entwickler haben oft keinen Zugriff auf die Live-Systeme. Das heißt, dass die Administratoren das Deployment der Anwendungen vornehmen.

Hier hilft einem Rex::Apache::Deploy. Das Modul besteht aus 3 Untermodulen:

Rex::Apache::Build
Dieses Modul stellt einem Funktionen zum Erstellen eines Applikationsarchivs zur Verfügung.

Rex::Apache::Inject
Dieses Modul enthält Funktionen um Konfigurationsdateien innerhalb eines Applikationsarchivs zu manipulieren.

Rex::Apache::Deploy
Dieses Modul enthält verschiedene Deploy-Mechanismen.

Das Applikationspaket schnüren

Das Modul Rex::Apache::Build bietet die Möglichkeit, eine Anwendung in ein tgz-Archiv (*.tar.gz) zu verpacken um es dann z.B. auf einen Installations-Server hochzuladen.

Der folgende Code (Task *build*) liest die Datei *lib/WebApp.pm* aus um die Version zu bekommen und erstellt dann das Archiv *webapp-{version}.tar.gz*. Der Task *upload* lädt das Archiv dann auf den Server *installsrv01* hoch.

```
get_version_from "lib/WebApp.pm",
     qr{\$VERSION = "([^"]+)";};

task "build", sub {
   build "webapp";
};

 task "upload", "installsrv01", sub {
   my $version = get_version;
   upload "webapp-$version.tar.gz",
          "/space/packages";
};
```

Die Applikation ausrollen

Das Deployment einer Anwendung ist genau so einfach wie das Paketieren. Rex kennt aktuell zwei Deployment-Mechanismen.

Symlink

Hier wird das Paket auf den Zielserver hochgeladen, entpackt und danach wird vom DocumentRoot des Webservers ein symbolischer Link zu der gerade entpackten Anwendung erstellt. Damit besteht die Möglichkeit einfach und schnell Rollbacks durchzuführen falls ein Deployment fehlschlagen sollte.

Tomcat

Wie der Name schon sagt, kann man mit diesem Modul Tomcat-Anwendungen ausrollen.

```
 1 use Rex::Apache::Deploy "Symlink";
 2
 3 group www => "www01", "www02";
 4
 5 deploy_to "/var/www";
 6 document_root "/var/www/html";
 7
 8 get_version_from "lib/WebApp.pm",
 9       qr{\$VERSION = "([^"]+)";};
10
11 task "deploy", group => "www", sub {
12    deploy "webapp";
13 };
```

Das ist die einfachste Form eines Deploy-Tasks. Dieser Task lädt das Paket, das vorher mit dem Task *build* erstellt wurde auf die Server *www01* und *www02* hoch und entpackt es in das Verzeichnis */var/deploy/{version}*.

Soll sichergestellt werden, dass beim Deployment keine Fehler aufgetreten sind, besteht die Möglichkeit einen eigenen Task *test* zu erstellen. Dieser Task kann nach dem Deployment ausgeführt werden um bei einem fehlerhaften Deployment auf die alte Version zurückzurollen (siehe Listing 1).

Verwalten mehrerer Environments

In einem gut organisierten Entwicklungs- und Betriebsteam durchläuft ein Release immer mehrere Systeme. Angefangen von einem Integrationsystem über ein Stagingsystem, gegebenenfalls noch andere Systeme bis zum Livesystem.

Damit man für das Deployment auch das gleiche Rexfile verwenden kann, kennt Rex sogenannte Environments. In einem Environment werden Servergruppen definiert oder auch spezielle Tasks, die nur in einem bestimmten Environment verfügbar sein sollen (siehe Listing 2).

```
 1 use Rex::Transaction;
 2
 3 task "deploy", group => "www", sub {
 4    my $old_live = get_live_version;
 5
 6    transaction {
 7
 8       on_rollback {
 9       say "Rolle zurueck auf $old_live\n";
10          switch_to_version($old_live);
11       };
12
13       deploy "webapp";
14
15       needs "test";
16
17    };
18 };
19
20 task "test", sub {
21   require LWP::Simple;
22   my $server =
23   Rex::get_current_connection->{"server"};
24
25    my $ret = LWP::Simple::get(
26          "http://$server/_system/check"
27     );
28    if($ret ne "OK") {
29       die
30    "Test der Applikation war fehlerhaft.";
31    }
32 };
```

Listing 1

```
 1 environment test => sub {
 2    group www => "test-www01";
 3    user "root";
 4    password "pass";
 5 };
 6
 7 environment stage => sub {
 8    group www => "stage-www01",
 9           "stage-www02";
10    user "root";
11    password "pass";
12 };
13
14 environment live => sub {
15    group www => "www01", "www02";
16    user "deploy";
17    password "pass";
18 };
19
20 task "deploy", group => "www", sub {
21    deploy;
22 };
```

Listing 2

Mit dem Parameter *-E* kann dann das gewünschte Environment geladen werden:

```
rex -E stage taskname
```

Ein komplettes Szenario

Wenn wir jetzt die einzelnen Schritte zusammenfassen bekommen wir ein Rexfile, das Entwickler verwenden können um sich eigene Testsysteme zu erstellen. Aber auch die Administratoren können es verwenden um die Liveumgebung zu installieren und auszurollen. Damit ist sichergestellt, dass Test- und Livesysteme immer im gleichen Zustand sind (siehe Listing 3).

Links

[1] http://rexify.org/compatibility.html

[2] http://rexify.org/get/

```
use strict;
use warnings;

use Rex::Apache::Build;
use Rex::Apache::Deploy "Symlink";

use Rex::Transaction;

user "root";
password "test";

group www => "www[01..02]";

desc "Webserver vorbereiten - Benoetigte Pakete installieren.";
task "prepare", group => "www", sub {

   install package => [
                        "apache2",
                      ];

   service "apache2", "ensure" => "started";

};

get_version_from "lib/WebApp.pm", qr{\$VERSION = "([^"]+)";};
desc "Applikationsarchiv erstellen.";
task "build", sub {
   build "webapp";
};

desc "Applikation den Admins zur Verfuegung stellen.";
task "upload", "installsrv01", sub {
   my $version = get_version;
   upload "webapp-$version.tar.gz", "/space/packages";
};

deploy_to "/var/www";
document_root "/var/www/html";
desc "Applikation auf Webserver deployen.";
task "deploy", group => "www", sub {

   my $old_live = get_live_version;
   transaction {

      on_rollback {
         say "Wechsle zur alten Version: $old_live";
         switch_to_version($old_live);
      };

      deploy "webapp";
      needs "test";

   };
};

desc "Test der Applikation.";
task "test", sub {
   require LWP::Simple;
   my $server = Rex::get_current_connection->{"server"};

   my $ret = LWP::Simple::get("http://$server/_system/check");
   if($ret !~ m/OK/i) {
      die("Test fehlgeschlagen!");
   }
};
```

Renée Bäcker

Ein CPAN für eigene Module

Das neue Projekt ist fertig, die Anwendung wurde modular entwickelt und es sind dabei einige Distributionen entstanden. Ein Teil davon kann auf CPAN geladen werden, ein anderer Teil ist zu speziell oder enthält Wissen über Interna des Unternehmens. Diese Module sollen dann nicht auf dem CPAN veröffentlicht werden. Trotzdem möchte man die Vorteile eines CPAN nutzen - vor allem für die Installation von Modulen über die CPAN-Clients wie *CPANPLUS*, *cpanminus* oder *CPAN.pm*.

Dann wird es Zeit für ein CPAN, in dem eigene Module bzw. Distributionen verwaltet werden können. Dieses Ziel kann auf mehreren Wegen erreicht werden. In diesem Artikel sollen ein paar dieser Wege gezeigt werden. Einige Möglichkeiten wurden erst vor kurzem entwickelt und haben daher noch einige Kinderkrankheiten.

Kritik an eigenen privaten CPANs kommt immer wieder auf und es heißt, dass man lieber direkt Pakete für die Paketmanager der Betriebssysteme zur Verfügung stellen sollte, weil man damit auch Abhängigkeiten zu Nicht-Perl-Bibliotheken berücksichtigen kann. Für mich ist das aber nicht wirklich wichtig und ich nutze lieber die CPAN-artige Infrastruktur. Denn damit muss ich nicht für jeden Kunden eigene Pakete bauen, nur weil ein anderes Betriebssystem zum Einsatz kommt.

Damit ein eigenes Repository von den CPAN-Clients genutzt werden kann, sind folgende Bedingungen zu erfüllen:

• Index-Dateien zur Verfügung stellen, die Informationen darüber enthalten, welches Modul in welcher Distribution zu finden ist. Diese Dateien sind *02packages.details.txt.gz* und *03modlist.data.gz*.

• Eine bestimmt Ordnerstruktur, die weiter hinten im Artikel noch zu sehen ist.

• Die CPAN-Clients müssen in der Lage sein, Pakete herunterzuladen. Das geht über `http://`, `ftp://` und `file://`.

Die nachfolgenden Module erledigen alles schon automatisch, so dass man sich darum keine Gedanken mehr machen muss.

Ein kleines CPAN und meine Module

Schon vor längerer Zeit haben wir in diesem Magazin `CPAN::Mini` vorgestellt. Damals aber als Lösung für ein anderes Problem - Module installieren ohne Internetverbindung. Mit `CPAN::Mini` werden immer die aktuellsten Versionen auf der lokalen Platte abgelegt. Beim ersten Start dauert es natürlich eine ganze Zeitlang bis die Module von CPAN heruntergeladen werden. Alle weiteren Läufe sollten relativ flott gehen (wenn man nicht gerade nur einmal im Jahr die aktuellsten Versionen holt).

Auf `CPAN::Mini` aufbauend wurde `CPAN::Mini::Inject` entwickelt, mit dem die eigenen Module in das Mini-CPAN integriert werden können. Das Modul kommt auch gleich mit einem Skript, das man zum Einbinden der eigenen Module in das Repository benutzen kann - *mcpani*. Leider muss man beim Hinzufügen von Modulen sehr viele Angaben selbst machen:

```
mcpani --add --module Test::Distro \
   --authorid RENEEB --modversion 1.24 \
   --file ./Test-Distro-1.24.tar.gz
```

Hier wäre es wünschenswert, wenn mehr Daten aus dem Modul gezogen werden würde.

mcpani benötigt eine Konfigurationsdatei, die typischerweise unter *~/.mcpani/config* gespeichert ist:

```
local: /www/CPAN
remote: ftp://ftp.cpan.org/pub/CPAN
repository: /work/mymodules
passive: yes
dirmode: 0755
```

Die ersten beiden Einträge geben an, welches Verzeichnis dann als CPAN-Spiegel dient und von welcher Seite die Module kommen. Im `repository` werden die Module, die man selbst hinzufügt, gespeichert. Das ist dann wichtig, wenn man das MiniCPAN mit

```
mcpani --update
```

aktualisiert. Denn dann werden erst die CPAN-Module auf den neuesten Stand gebracht und dann werden die eigenen Module wieder zum MiniCPAN hinzugefügt.

Wenn nur die CPAN-Module aktualisiert werden sollen, reicht ein `mcpani --mirror`.

Man braucht ja nicht unbedingt alle (aktuellen) Module vom CPAN. Das sind ein paar hundert Megabyte an Daten. Leider hat `CPAN::Mini::Inject` das nicht unterstützt. Wenn man also nur seine eigenen Module in einem Repository haben möchte, muss man auf eine andere Lösung zurückgreifen. Eine der möglichen Lösungen wird in den nachfolgenden Abschnitten beschrieben - `Pinto`.

Pinto

Ein weiteres ganz neues Modul ist `Pinto`, mit dem auch verschiedene Repositories verwaltet werden können. Vielleicht sind die cpXXXan-Seiten von David Cantrell bekannt. Dort findet man auch für ältere Perl-Versionen noch Module, die dort funktionieren (z.B. *http://cp5.6.2an.barnyard.co.uk/*). Gerade wenn man Perl 5.6.x oder gar noch älter verwendet, hat man mit dem normalen CPAN-Index häufig Pech, dass die Module mit den Perl-Versionen nicht mehr funktionieren. Woher aber wissen, welche Version des Moduls noch mit der Perl-Version funktioniert hat? Die Ideen von David Cantrell sind in die Entwicklung von `Pinto` eingeflossen. Umgesetzt wird `Pinto` von Jeffrey Thalhammer, der schon die ersten Versionen von `Perl::Critic` entwickelt hat.

Mit Pinto ist es ebenfalls möglich, mehrere Repositories einzurichten und zu verwalten. Allerdings steht das Modul noch ganz am Anfang der Entwicklung und es sind noch ein paar Kinderkrankheiten da, aber es ist ein guter Ansatz und für kleinere Repositories durchaus schon zu gebrauchen.

Ich möchte hier zeigen, wie ich in Zukunft meinen Kunden Perl-Pakete zur Verfügung stellen werde.

Ein Repository pro Kunde

In Abbildung 1 ist zu sehen, wie es aussehen wird: Jeder Kunde bekommt ein Repository, in dem die Pakete liegen, die für das jeweilige Projekt benötigt werden. Dort werden aber nicht nur die eigenen Module gespeichert, sondern auch die jeweils bei der Entwicklung zum Einsatz gekommenen Versionen der CPAN-Module. Damit soll gewährleistet werden, dass die Anwendung schnell neu aufgesetzt werden kann - ohne dass die Versionen der CPAN-Module erst noch herausgesucht und von BackPAN heruntergeladen werden müssen.

Der Kunde kann dann über die Konfiguration des CPAN-Clients sein Repository einbinden, so dass er ohne großen Aufwand die Module installieren kann.

Pinto nutzen

Als erstes müssen Repositories angelegt werden. Das `Pinto`-Paket liefert gleich mehrere Programme mit und eines davon ist `pinto-admin`. Mit diesem Programm werden die Repositories verwaltet.

```
pinto-admin --repos=/var/pinto/reposA create

reneeb@ubuntu:~$ sudo pinto-admin \
        --repos=/var/pinto/reposA create
Created new repository at directory \
        /var/pinto/reposA
```

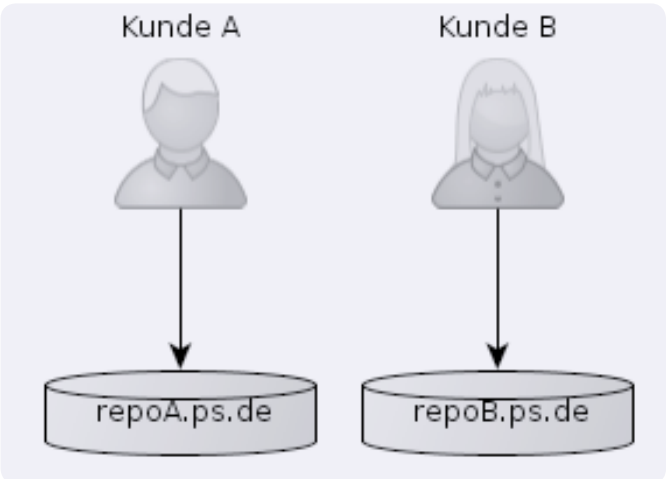


Abbildung 1: Pro Kunde gibt es ein Repository

Der nächste Schritt ist das Hinzufügen von Module zum Repository. Auch hierfür wird `pinto-admin` verwendet.

```
pinto-admin --repos=/var/pinto/reposA add \
    Nexmo-SMS-0.06.tar.gz
```

Sobald die Distribution zum Repository hinzugefügt wird, werden die Index-Dateien geschrieben wie auf CPAN:

```
reneeb@ubuntu:/var/pinto/reposA$ tree
.
├── authors
│   ├── 01mailrc.txt.gz
│   └── id
│       └── R
│           └── RE
│               └── RENEEB
│                   ├── CHECKSUMS
│                   ├── Nexmo-SMS-0.06.tar.gz
│                   └── Nexmo-SMS-0.07.tar.gz
└── modules
    ├── 02packages.details.txt.gz
    └── 03modlist.data.gz
```

Ich habe schon angesprochen, dass ich in den Repositories auch Module vom CPAN verwalten möchte. Um hier nicht den Umweg über das Herunterladen des Pakets und dann das Hinzufügen in das Repository gehen zu müssen, kennt `pinto-admin` einen Befehl, mit dem das Modul direkt von einem CPAN-Spiegel in das Repository geladen wird.

```
pinto-admin --repos=/var/pinto/reposA \
    import Plack
```

Beim Import des Pakets werden auch alle Abhängigkeiten heruntergeladen. Das ist sehr praktisch, weil dann im Repository genau die Module vorgehalten werden, die auch bei der Entwicklung zum Einsatz gekommen sind. Man hat also immer einen definierten Stand an Paketen.

Die Arbeit im Team und von unterwegs

Manche Module werden ja nicht von einer einzelnen Person entwickelt, sondern von mehreren Entwicklern im Team. Der Quellcode an sich wird in der Regel in einem Versionskontrollsystem wie *git* oder *svn* verwaltet (bei wem das nicht der Fall ist, sollte sich solche Systeme nochmal anschauen). Aber was passiert dann? Wenn sich immer nur einer um die Repositories kümmert, entsteht dort ein Flaschenhals. Die Person kann krank werden, das Unternehmen verlassen oder einfach keine Zeit haben.

Deshalb ist es notwendig, dass jedes Teammitglied die Distributionen in die Repositories einspielen. Ein Weg - mit `Dist::Zilla` - wird noch weiter unten gezeigt.

Damit aber die Teammitglieder auch von überall mit den Repositories arbeiten können, gibt es die Programme `pinto-remote` und `pinto-server`. Wie man sich vielleicht schon vom Namen her denken kann, ist `pinto-server` die Serverkomponente. Mit entsprechenden Clients - wie z.B. das `pinto-remote` - können dann auch von entfernten Rechnern aus Distributionen zum Repository hinzugefügt oder auch wieder gelöscht werden.

Als erstes muss der Server gestartet werden:

```
pinto-server --repos=/path/to/repository
```

Standardmäßig lauscht der Server auf Port 3000. Wer einen anderen Port nutzen möchte, kann das über den Parameter `--port` einstellen. Wenn der Server als *Daemon* laufen soll, muss man das Flag `--daemon` setzen.

Leider ist es mit dem Server-Programm nicht möglich, mehrere Repositories mit einem Prozess zu verwalten.

Sobald der Server gestartet ist, kann man mit dem Client loslegen. Eine Distribution fügt man wie folgt in das Repository ein:

```
pinto-remote --repos=http://repo.server \
    add Distribution-0.01.tar.gz
```

Revisionssicherheit des Repositories

Code liegt in der Regel in einem Versionskontrollsystem. Mit `Pinto` kann man auch das gesamte Repository selbst in einem solchen System halten. Damit kann man Änderungen am Repository auch wieder zurücknehmen. Unterstützt werden von Haus aus *Git* und *SVN*.

Um dieses Feature nutzen zu können, muss beim Erzeugen des Repositories angegeben werden, dass es mit einem solchen Versionskontrollsystem verwaltet werden soll. Dazu gibt es den Parameter `--store`.

```
pinto-admin --repos=/var/pinto/name
    \create --store=Pinto::Store::VCS::Git
```

Unter Windows wird das mit Git aber nicht richtig funktionieren, weil `Pinto` dafür das Modul `Git::Repository` verwendet und das hat unter Windows noch ein paar Bugs. Als Alternative kann man aber auch das SVN-Plugin (`Pinto::Store:VCS::Svn`) für das Repository nehmen. Für die SVN-Unterstützung muss das Programm *svn* im Pfad liegen.

Auch nachträglich kann ein Pinto-Repository noch unter Versionskontrolle gestellt werden. Dazu muss in der Datei *pinto.ini*, die im Pinto-Repository unter `/.pinto/config/` liegt, das Git-Modul von Pinto als Speichermodul eingetragen werden.

```
# In /pfad/repos/.pinto/config/pinto.ini
store = Pinto::Store::VCS::Git
```

Danach werden die ganz normalen `git`-Befehle verwendet:

```
# Pinto-Repository als Git-Repository
# initialisieren
$> cd /pfad/repos
$> git init

# Alle initialen Dateien und Verzeichnisse
# hinzufügen
$> git add .pinto authors modules
$> git commit -a -m 'New Pinto repos'
```

```
<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  ServerName  kundeA.repo.perl-services.de

  DocumentRoot /var/pinto/reposA/

  <Directory />
     Options FollowSymLinks
     AllowOverride None
  </Directory>
  <Directory /var/pinto/reposA/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    allow from all
  </Directory>

  ErrorLog /tmp/error.log
  LogLevel warn

  CustomLog /tmp/access.log combined
</VirtualHost>
```

Listing 1

Zugriff für die Kunden

Bisher können die Kunden noch nicht wirklich die Repositories nutzen, da ein Zugriff von außerhalb nicht möglich ist. Jetzt kommt ein Webserver ins Spiel. Da schon ein Apache 2 läuft, wird auch für diese Aufgabe der Indianer herangezogen. Es besteht auch die Möglichkeit mit dem oben erwähnten `pinto-server`-Programm das Repository für Kunden nutzbar zu machen, aber hier soll mal gezeigt werden, wie das mit dem Apachen aussieht.

Für jedes Repository wird ein *VirtualHost* erzeugt (Listing 1).

Danach muss das Repository noch im CPAN-Client eingetragen werden, bzw. bei `cpanminus` kann man mittels `--mirror` direkt die URL angeben. Aber wie es bei `CPAN.pm` aussieht, ist in Listing 2 gezeigt.

Direkt in das Repository releasen

Gerade wenn Module in mehreren Projekten zum Einsatz kommen ist es recht umständlich, eine neue Version eines Moduls in jedes einzelne Repository einzuspielen. Es ist immer die gleiche Vorgehensweise und man vergisst doch mal schnell ein Repository. Aus diesem Grund gibt es ein Plugin für `Dist::Zilla`, das bei einem Release die neue Version in die konfigurierten Repositories schiebt (Abbildung 2). Das Plugin heißt `Dist::Zilla::Plugin::Pinto::Add` und ist auf CPAN zu finden.

In der *dist.ini* des zu releasenden Moduls ist dann folgendes zu finden:

```
[Pinto::Add]
repos = /var/pinto/reposA
repos = /var/pinto/reposB
author = RENEEB
```

```
cpan[1]> o conf urllist push http://kundea.repo.perl-services.de/
cpan[2]> ls RENEEB
Fetching with LWP:
http://kundea.repo.perl-services.de/authors/id/R/CHECKSUMS
Fetching with LWP:
http://kundea.repo.perl-services.de/authors/id/R/RE/CHECKSUMS
Fetching with LWP:
http://kundea.repo.perl-services.de/authors/id/R/RE/RENEEB/CHECKSUMS
    7067 2009-11-22 RENEEB/Bio-FASTASequence-0.06.tar.gz
   27047 2011-10-27 RENEEB/FabForce-DBDesigner4-0.31.tar.gz
   20342 2011-12-02 RENEEB/FabForce-DBDesigner4-DBIC-0.0803.tar.gz
   12858 2012-01-02 RENEEB/Nexmo-SMS-0.06.tar.gz
   12861 2012-01-02 RENEEB/Nexmo-SMS-0.07.tar.gz
   40517 2012-01-02 RENEEB/Nexmo-SMS-0.08.tar.gz
   10018 2011-04-09 RENEEB/Test-CheckManifest-1.24.tar.
```

Listing 2

Damit sieht dann die Ausgabe beim Release-Prozess von `Nexmo::SMS` wie in Listing 3 dargestellt.

Um mehrere Repositories angeben zu können, ist das gepatchte Modul auf meinem Github-Account (*http://github.com/reneeb*) notwendig. Mit etwas Glück ist der Patch bis zum Erscheinungsdatum dieser Ausgabe auch schon in die CPAN-Distribution eingeflossen.

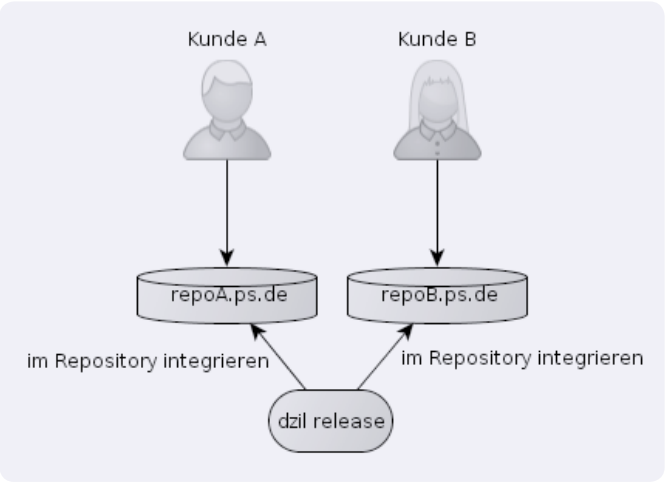


Abbildung 2: Beim Release wird das Modul in die Repositories hinzugefügt

Fazit

Mit den hier gezeigten Modulen ist es einfach möglich, eigene CPAN-artige Repositories für seine Kunden zu erstellen. Die Kunden können dann mit der gewohnten Werkzeugkiste arbeiten wenn es an die Installation der Distributionen geht.

Jede Lösung hat seine Vor- und Nachteile, aber für mich persönlich ist `Pinto` die bessere Alternative, auch wenn es hierfür (noch) keine Weboberfläche gibt, über die die Kunden nach Distributionen suchen und sich die Dokumentation anschauen können.

```
reneeb@ubuntu:~/perl-Nexmo-SMS$ dzil release
[DZ] beginning to build Nexmo-SMS
[DZ] guessing dist's main_module is lib/Nexmo/SMS.pm
[VersionFromModule] dist version 0.07 (from lib/Nexmo/SMS.pm)
[DZ] extracting distribution abstract from lib/Nexmo/SMS.pm
[DZ] writing Nexmo-SMS in Nexmo-SMS-0.07
[DZ] building archive with Archive::Tar; install Archive::Tar::Wrapper for improved speed
[DZ] writing archive to Nexmo-SMS-0.07.tar.gz
[Pinto::Add] checking if repository at /var/pinto/reposA is available
[Pinto::Add] checking if repository at /var/pinto/reposB is available
[Pinto::Add] adding Nexmo-SMS-0.07.tar.gz to repository at /var/pinto/reposA
[Pinto::Add] added Nexmo-SMS-0.07.tar.gz ok
[Pinto::Add] adding Nexmo-SMS-0.07.tar.gz to repository at /var/pinto/reposB
[Pinto::Add] added Nexmo-SMS-0.07.tar.gz ok
```

Listing 3

Perl Weekly

Sie sind damit beschäftigt, am laufenden Band Code zu schreiben oder die Entwickler zu beschäftigen. Sie interessieren sich für Perl, haben aber keine Zeit, um durch Dutzende und Hunderte von Artikeln und Blog-Posts jeden Tag zu gehen.
Sie wollen ein Auge auf die Entwicklung von Perl haben, ohne in einem Meer von Blog-Posts zu ertrinken. Sie brauchen jemanden, der Sie auf die wichtigsten Nachrichten und Artikel in der Perl-Welt aufmerksam macht.

"Vielen Dank! Exzellente Zusammenstellung und ein guter Weg, um zu sehen, was los ist."
Job van Achterberg

"Danke für die Zusammenstellung des Newsletters. Es ist eine hilfreiche Ressource."
Richard Dice

"Ich genieße die Zusammenfassungen von Artikeln in @ szabgab Perl-Weekly. Mehr Perl news = gute Nachrichten."
Andy Lester

Jeden Montag gibt es die Zusammenfassung der interessantesten Nachrichten und Artikel im

PerlWeekly Newsletter

Melden Sie sich noch unter http://perlweekly.com an!

Herbert Breunung

WxPerl Tutorial - Teil 9: Mächtige Widgets 1

Hiermit beginnen die abschließenden Teile dieses umfassenden Tutorials. Sämtliche Grundlagen wurden gelegt, um vollständige Programme zu schreiben. Das waren vor allem Dinge, die von einem graphischen Rahmenwerk erwartet werden. Doch Wx bietet mehr. Manches davon kann man mit dem *CPAN* im Rücken lächelnd übergehen. Anderes (wie etwa `Wx::STC`) war für den Autor ursprünglich der Grund, sich für Wx zu entscheiden. Dieser Teil wird mit einer Gesamtschau über alle Widgets beginnen, was vor allem nützlich ist, weil eine zunehmende Anzahl im CPAN verstreut ist. Darauf werden XML und HTML Thema sein, welche in Wx eine große Rolle spielen - auch um Oberflächen wie Webseiten oder mit graphischen Baukästen zu entwerfen. Der Schlussteil wird das Erzeugen eigener Widgets beleuchten.

Doch zuvor noch eine Bemerkung zur aktuellen Entwicklung. Im Teil 6 wurde der Begriff Applikation verwendet, der 2011/12 eher mit angebissenen Äpfeln und kleinen grünen Robotern in Verbindung gebracht wird. Bereits früh wurde Wx auf *Symbian* und *Windows Mobile* portiert, so dass Wx angepasste Klassen und Methoden hat, um die spezielle Steuerung auf kleinen Bildschirmen zu ermöglichen. Mittlerweile werden darüber hinaus *iPhone SDK* und *embedded GTK+* unterstützt. Nur gibt es leider derzeit noch keine Berichte von mobilen Wx-Apps.

Übersicht

Die Mehrzahl der Widgets hat eine einfach verständliche Schnittstelle, die nur leider oft zu wortreichem Code nötigt. Daher widmet sich dieser Teil eher den Ecken und Kanten der Widgetverwendung. Denn oft reicht der Start des Skriptes `wxperl_demo` (auf *nixes mit Endung *.pl*, unter Windows zusätzlich mit oder ohne Endung *.bat*) und man sieht bereits in Aktion und mit Beispiel was gebraucht wird. Leider ist das im Modul `Wx::Demo` enthaltene Skript nicht ganz vollständig. Die dekorative `AnimationCtrl` würde man zum Beispiel vergebens suchen. Sie spielt ununterbrochen einen Film ab (eine `Wx::Animation`) und hat nichts mit dem zu tun, was man von *youtube* kennt. Dazu bräuchte es die ebenfalls nicht enthaltene `Wx::MediaCtrl`. Manchmal sind es nur die Namen, die etwas irreführen. Hierfür ist die deutsche Liste aller Widgets, Ereignisse und Konstanten *http://wiki.perl-community.de/Wissensbasis/WxPerlTafel* der Wegweiser.

Hoffentlich sieht das geplante *Docular* in diesem Jahr das Tageslicht, um endlich alle Methoden, Konstanten und Ereignisse je Klasse überblicken zu können. Da WxPerl nicht immer alles per XS weiterleitet, wären verlässliche Angaben äußerst wünschenswert.

Widgetologie

Oft reicht es schon, die Signalworte zu kennen, um vom Namen den Zweck abzuleiten. Einige wurden bereits in einzelnen Teilen erklärt, trotzdem wäre eine Zusammenfassung der meisten hilfreich. *Window* hat in diesem Kontext wenig Aussagekraft. Es steht seltener für Fenster, öfters für Widget verschiedener Art, meist welche, die andere Widgets kontrollieren (was Fenster tun). Immerhin hieß WxWidgets vor Microsofts Einspruch WxWindows und `Wx::Window` ist immer noch die Basisklasse für alles was sichtbar ist.

Im Gegensatz dazu ist eine *Control* (von `Wx::Control` abgeleitet) etwas, das auf Nutzereingaben reagiert und was meist mit einem Widget gemeint ist. Menüs und Menüleisten, welche überall angebracht werden können, zählen nicht dazu, weil hier das Reagieren auf Mauskoordinaten an das OS abgegeben wird. *Ctrl* selbst tragen die eher komplexeren Widgets im Namen. Gibt es 2 Widgets mit gleichem Stamm-

namen, bietet die *Ctrl* mehr Optionen. Die `PickerCtrl` wurden im Teil 2 mit den Dialogen behandelt, da sie einen gleichnamigen Dialog rufen, welcher die Auswahl abfragt, die sie anzeigen.

Auf "book" endende Widgets kontrollieren eine Reihe Panel, von denen nur eines zugleich sichtbar sein kann. Die Reiterleiste (Notebook) ist die bekannteste. Sie wurden in Teil 4 erklärt.

Enthält der Name ein "Box", geht es um Widgets in einem sichtbaren Rahmen, meist eine eindimensionale Elementeliste. Ein eingeschobenes V deutet an, dass bei genügend Elementen, eine vertikale Skrollleiste einspringt. Manche Widgets weichen da lieber auf mehrere Spalten aus. "Combo" deutet auf Widgets die aus Mehreren zusammengesetzt sind.

Einfache Widgets

Nun endlich zu den Widgets, mit den einfachsten beginnend: Die wichtigsten passiven, dekorativen Widgets wurden im vierten Teil vorgestellt. Diese beginnen im Namen meist mit "Static", was lediglich bedeutet, dass der Benutzer sie nicht ändern kann, das Programm jederzeit. Zu ihnen ließe sich ebenfalls `AnimationCtrl` und der Fortschrittsbalken zählen, der im Gegensatz zum `ProgressDialog` die wenig intuitive Bezeichnung `Wx::Gauge` trägt. Die API ist sehr schlicht. Zudem ist nur darauf zu achten, dass der dritte Parameter des Konstruktors den Maximalwert der Skala bekommt, die hochgezählt wird. Im Callback des passenden *Event* muss dann nur ein triviales `$gauge->SetValue($x);` stehen.

Oft lässt sich im Voraus schwer ermessen wie lange etwas dauert und man möchte nur eine Animation zeigen. Mit dem zusätzlich zu installierenden Modul `Wx::Perl::Throbber` lässt sich, wenn nötig, mit nur einer Zeile Code und einigen `Wx::Bitmap` ein Film erstellen, der sofort abspielbereit ist und auf eine beliebige Fläche projiziert wird.

```
my $throbber = new Wx::Perl::Throbber(
    $window, -1, \@bitmap,
    [230,40], [132, 32], 200, undef, undef,
    'Bitte fröhlich bleiben'
);
$throbber->Start; # später Rest oder Stop
```

Verständlicher (im Sinne von deklarativ) wäre aber:

```
my $throbber = new Wx::Perl::Throbber
    ($window, -1, \@bitmap, [230,40],
      [132, 32]);
$throbber->SetLabel
    ('Bitte fröhlich bleiben');
$throbber->ShowLabel(1);
$throbber->SetFrameDelay(200);
```

Das Beste ist jedoch: Nach einem `$throbber->Destroy;` ist alles wieder aufgeräumt.

Bereits im zweiten Teil wurden die Möglichkeiten des `Wx::Button` gezeigt. Knöpfe mit graphischer Oberfläche sind die `Wx::BitmapButton`, welche beim Erzeugen an dritter Stelle eine *Bitmap* (sechster Teil) statt Text annehmen, der Rest bleibt gleich.

Mehr Auswahl

Auch die `Checkbox` und `Radiobutton`, sowie `Radioboxen` kamen im vierten Teil an die Reihe, jedoch nicht die `CheckListBox`. Diese vereint eine Liste `Checkboxen` in einem Widget mit Rahmen. Genauso vereint eine `RadioBox` eine Liste `Radiobutton`, nur mit Überschrift. Das spart beträchtlich Aufwand, bringt dafür Kontrollverlust. `CheckListBox` bietet nicht die Einstellmöglichkeiten einer `Radiobox`:

```
Wx::RadioBox->new(
    $panel, -1, 'Überschrift',
    [-1,-1], [-1,-1], [1..15],
    2, &Wx::wxRA_SPECIFY_COLS
    # alternativ wxRA_SPECIFY_ROWS
);
```

In dem Beispiel sind die 15 Optionen über zwei Spalten verteilt. Eine `CheckListBox` kennt nur eine Spalte. Dafür passt sich die `CheckListBox` dem vorhandenen Platz an und verleiht sich selbst - wenn nötig - eine Scrollleiste. Um gleiches für eine `RadioBox` zu erhalten, müsste man sie mit einem `VScrolledWindow` unterlegen. Hat die `CheckListBox` nur wenige Elemente, kann in ihr unten ein auffallender Freiraum bleiben, den `Fit` nicht behebt, sondern nur Handarbeit. Das bedeutet per `GetFont` und `GetPointSize` die Höhe je Element berechnen und per

```
$sizer->SetItemMinSize(
    $widget, $breite , $hoehe
);
```

dem Sizer sagen auf welche Höhe er das Widget zu halten hat. Ein SetSize würde nur bis zu nächsten Größenänderung des Fensters halten.

Während `Checkbox` und `Radiobutton` einheitlich mit `SetValue` in ihrem Zustand versetzt werden, ist es bei der `Radiobox` `SetSelection`, was bei der `CheckListBox` nur den Caret (Tastaturcursor) setzt. Die Häkchen setzt dann *Enter*, die Leertaste oder die Methode `Check`, welche die Nummer der `Checkbox` und ihren kommenden Zustand benötigt. Das ist wohl dem Umstand geschuldet, unbedingt die Windows-eignen Checkboxen zu unterstützen, welche drei Zustände haben können, normalerweise reicht 0 und 1.

Schön an der `CheckListBox` ist die volle Freiheit über die Liste der Label. Sie kann mit `Set` (bekommt eine *ArrayRef*) vollständig neu gesetzt werden, oder einzeln verändert `SetString($n, $label)` oder erweitert `Insert($label, $n)` werden. Das riecht nach Inkonsistenz, ist aber verständlich, weil der manchmal einzige Parameter `$n` durchgängig die erste Position hat. Fast die gesamte API funktioniert über die Nummer (Index beginnt mit 0) des *Items*. Die Ausnahme war notwendig, da `$nr` in dem Fall optional ist. Ein Wunsch auf Einfügepositionen macht bei sortierten Listen keinen Sinn. `Insert` entstammt nämlich der Klasse `Wx::ControlWithItems`, aus welcher die Arrayfunktionen für die Auswahlwidget `Radiobox`, `CheckListBox`, `Choice`, `ListBox`, `ComboBox` und `BitmapComboBox` stammen. Ihnen ist dieser Abschnitt gewidmet.

Von den verbleibenden vier bieten drei ihre Auswahl über ein `PopupWindow` (Kontextmenü - siehe Folge 6) an, was die Platz sparendste Alternative ist. `Wx::Choice` ist dabei die einfachste Variante und liefert soweit nichts Neues. Die Befehle zur Anzahl der Spalten haben nur unter *Motif* einen Effekt und dienen meist nur der Verwirrung. `Choice` sollte man nehmen, wenn eine Wahl aus einer fest stehenden Menge zu treffen ist. Darf der Nutzer eigene Wünsche anmelden, muss es eine `ComboBox` sein. Das ist in Wirklichkeit ein Texteingabefeld (`TextCtrl`) mit angeflanschtem Menü (`ComboPopup`). Soll dieses Bilder beinhalten, nimmt man selbsterklärend `BitmapComboBox`. Trotz ihres gleichen Namens sind beides ungleiche Schwestern. Während die meisten Plattformen erstere als Bestandteil kennen, ist die zweite ein von Wx nachempfundenes Imitat, was besonders sichtbar wird, wenn man das Betriebsystem seinen farblichen Vorlieben angepasst hat. Andererseits verbieten manche Plattformen wie GTK an einer `ComboBox` etwas optisch zu ändern. Windows ist in solchen Fragen oft die gutmütigste Umgebung und soll das Programm sich überall gleich verhalten, so kann auch Wx keinen Test ersetzen. Der ergonomische Nachteil der `ComboBox` sind die Maße des Popup-Knopfes. Die `Choice` hat den Vorteil, dass das gesamte Widget als Popup-Knopf fungiert.

Eine ähnliche Annehmlichkeit hat die "ständig ausgeklappte" `ListBox`, da sich ihre Elemente über die ganze Zeile erstrecken, nicht nur den Bereich des Textetiketts wie bei der `ListCtrl`. Während sich die `CheckListBox` aufplustert, macht sie sich manchmal klein (fordert vom Sizer weniger Höhe als sie braucht um alle Elemente anzuzeigen), lässt sich allerdings auch bei Platznot skrollen.. Mit `SetFirstItem` wird der sichtbare Bereich verstellt. Das macht aber nur Sinn wenn es nach einem `Select` oder `Deselect` geschieht, welches das angefasste Element ins Sichtbare verschiebt. Die API beider Widgets ist beinah identisch, da `CheckListBox` von `ListBox` erbt, was auch namentlich ablesbar ist. Der Unterschied besteht darin, dass man in der `CheckListBox` mehrere Elemente auswählen kann (`Check`, `IsChecked`), während bei der `ListBox`, ganz wie bei der `RadioBox`, die *Selection* bereits die Auswahl ist.

Abschließend sei zu den `ControlWithItems` angemerkt, dass `FindString` (Gib mir Nummer des Elements mit dem *label*!) und `SetStringSelection` (Ich wähle das *Label* mit dem Text!) wie `eq` arbeiten. Wer *Regex*-Magie benötigt, muss ein `grep` über die von `GetStrings` gelieferte Liste veranstalten.

Kleine Vorschau

An dieser Stelle berührt das Tutorial den nächsten Teil, da es Auswahlwidgets gibt, die *HTML* nutzen. Deswegen ist für alles Folgende `use Wx::Html;` Voraussetzung. In einer vollständigen Widget-Übersicht sollte ebenso `HtmlWindow` nicht fehlen.

```
my $hw = Wx::HtmlWindow->new($panel, -1);
$hw->SetPage("<html><body...>"); # oder:
$hw->LoadFile('/pfad/zur/datei.html');
```

Es kann einfaches *HTML* anzeigen (keine *style-tags*, *CSS* oder *JS*) und ist daher nicht mit *WebKit* zu vergleichen. Dafür kann es einiges, was über *HTML* hinausgeht und erlaubt damit Wx-

Programme in einem anderen Stil zu schreiben, der nächstes Mal vorgestellt wird.

Jetzt soll es um die `SimpleHtmlListBox` gehen, deren Erzeugung in das übliche Schema fällt.

```
use Wx::Html;
my $listbox = Wx::SimpleHtmlListBox->new(
    $parent, -1, [-1, -1], [-1, -1], $choices
);
```

`$choices` ist eine *ArrayRef* auf Strings die etwas `HTML` enthalten (ohne *html*- oder *body*-*Tags*). `GetSelection` und `SetSelection` kennt es noch, aber nur wenig mehr.

Die `HtmlListBox` ist wirklich weniger simpel, da man sie erst beerben muss, was in den Stoff der letzten Folge hineinragt. Der Aufwand lohnt sich nur, wenn die Liste sehr lang ist und das Nicht-Rendern nicht angezeigter *HTML*-Elemente einen ernsthaften Vorteil bringt. Nachdem der eigene Konstruktor den der Superklasse aufrief:

```
sub new {
    my( $class, @args ) = @_;
    my $self = $class->SUPER::new( @args );

    $self->SetItemCount( 1000 );

    return $self;
}
```

kündigt er an, dass 1000 Elemente erzeugt werden müssen. Tatsächlich wird die Methode `OnGetItem` aufgerufen, sobald das nächste Element sichtbar werden soll.

```
sub OnGetItem {
    my( $self, $nr ) = @_;
    return '<ul>....'
}
```

So ähnlich verhält es sich mit der `VListBox` und `ComboCtrl`. Es sind ebenfalls keine Instant-Widgets, die auf ihre Benutzung warten, sondern eher Baukästen oder Schablonen, die man beerben und erweitern muss. Doch das soll wirklich Inhalt des zwölften Teils werden.

Wenn man einfach nur einen Link setzten mag, geht das ganz ohne *HTML*.

```
my $link = Wx::HyperlinkCtrl->new
  ($parent, -1, 'wxperl',
     'http://wxperl.eu/');
```

Beim Klick auf den Link lassen sich weitere Arbeiten ausführen.

```
EVT_HYPERLINK($link, -1, sub{...} );
```

Im Callback bitte nicht `$event->Skip` vergessen, sonst muss man auch den Befehl zum Öffnen der Internetseite selbst geben.

```
Wx::LaunchDefaultBrowser
    ('http://wxperl.eu/');
```

Große Auswahl

Für gehobene Ansprüche der Wahl kennt Wx die `ListCtrl` und `TreeCtrl`. `ListCtrl` ist bei einfacher Nutzung nur dadurch von der `ListBox` zu unterscheiden, dass die Elemente in mehreren Spalten dargeboten werden. Erst bei umfangreichen Datensätzen können diese Widgets ihre Klasse ausspielen.

Wie bei komplexeren Widgets üblich, bekommt `ListCtrl` seine Daten nicht vom dritten (nach ID) oder fünften (nach Größe) Parameter des Konstruktors, sondern per eigener Methode. Deshalb ist die Erzeugung denkbar trivial.

```
my $list = Wx::ListCtrl->new(
   $parent, [-1, -1], [-1, -1], $style);
```

Solange kein Stil angegeben wird, lassen sich schnell ein paar Texthappen einfügen.

```
$list->InsertStringItem( $index, $label );
```

Sollen auch noch kleine bunte Bilder hinzukommen, braucht es den Stil `wxLC_ICON` oder `wxLC_SMALL_ICON`, je nachdem wie groß die Icon sein sollen. Dabei hat Wx keine wirkliche Ahnung, wie groß sie sind. Die beiden Konstanten sind nur *Handle* zu zwei `ImageList`. Diese Klasse zur Iconverwaltung lernten die Leser in Teil 4 kennen, als sie gebraucht wurden, um `Notebook`-Reiter zu bebildern. Auch hier müssen sie zuerst dem Widget zugewiesen werden.

```
my $small  = Wx::ImageList->new( 16, 16 );
my $normal = Wx::ImageList->new( 32, 32 );
...
$small->Add( Wx::Bitmap->new(
      $bild_datei_pfad, wxBITMAP_TYPE_ANY) );
...
$list->AssignImageList(
      $small, &Wx::wxIMAGE_LIST_SMALL );
$lit->AssignImageList(
       $normal, &Wx::wxIMAGE_LIST_NORMAL );
```

Bevor sie an die *Item* ausgehändigt werden.

```
# item mit bild und text
$list->InsertStringImageItem(
       $nr, "Label", $image_index );
# item nur mit bild
$list->InsertImageItem( $nr, $image_index );
```

Will man Farbe und Größe des Textes ändern, braucht es eine andere Strategie. Anstatt `InsertStringItem` muss man die Item als eigene Objekte erzeugen.

```
my $item = Wx::ListItem->new();
$item->SetText('label');
$item->SetImage($image_index);
$item->SetTextColour(...);
$item->SetBackgroundColour(...);
$item->SetWidth(...);
$item->SetFont(...);
$item->SetData( $data_ref );
$item->SetId(...);
$list->InsertItem( $item );
```

Obacht, dieses *Insert* entspricht einem *prepend* oder `unshift`, fügt also neue Elemente immer vorne an. Gut, die ID wird auch automatisch vergeben und Daten lassen sich an einfach erzeugte *Item* anhängen.

```
my $ID  = $lc->InsertStringItem(
                  $nr, "Label", );
$list->SetItemData( $ID, $data_ref );
```

Aber ansonsten ist dieser Weg reichlich umständlich, was auch zur Entstehung der `SimpleHtmlListBox` geführt haben mag. Ähnlich der `HtmlListBox` kennt auch die `ListCtrl` mit dem Stil `wxLC_VIRTUAL` einen Modus, in dem per `OnGetItemText`, `OnGetItemImage` und weiteren erst bei Bedarf die Elemente gebaut werden. Dies ist jedoch nur in Kombination mit `wxLC_REPORT` erlaubt, der eigentlichen Paraderolle des Widgets. Dabei gehört zu jedem Element eine Tabellenzeile. Also werden Spalten benötigt.

```
$list->InsertColumn(0, 'Tier');
$list->InsertColumn(1, 'Breite');
$list->InsertColumn(2, 'Gewicht');
```

Diese Spalten kommen nur im Reportmodus zum tragen und selbst dann kann die Titelzeile mit `wxLC_NO_HEADER` ausgeblendet werden. Die Informationen, die in den Spalten angezeigt werden, kommen aus den Daten der `ListItem`.

```
my $ID = $list->InsertStringItem(
                $nr, 'Walroß');
$list->SetItemData( $ID);
$list->SetItem( $ID, 1, '3 m' );
$list->SetItem( $ID, 2, '15 t' );
```

Der virtuelle Modus hat noch den zusätzlichen Vorteil, dass hier jede Angabe ein Icon bekommen darf. Es bedarf nur:

```
sub OnGetItemColumnImage {
    my( $liste, $zeile, $spalte ) = @_;
    return $icon_bitmap;
}
```

Praktisch sind auch die Stile `wxLC_SORT_ASCENDING` (sortier aufsteigend) oder `wxLC_SORT_DESCENDING` (sortier absteigend). Man muss nur einen Sortiercallback abliefern, was für Perlprogrammierer eine gewohnte Sache ist. In `$_[0]` und `$_[1]` (entsprechen `$a` und `$b`) sind die Nummern der *Item* (Zeilen), die gerade verglichen werden. Um die angedeutete Tierliste nach Gewicht (schwer zuerst) zu sortieren, braucht es:

```
$list->SortItems( sub {
    $list->GetItem($_[0])->GetColumn(2)
    >
    $list->GetItem($_[1])->GetColumn(2)
} );
```

Auch editierbare Etiketten (Stil `wxLC_EDIT_LABELS`) oder eine einfache Auswahl (`wxLC_SINGLE_SEL` - immer nur ein aktives Element) können sehr nützlich sein. Zusätzliche Linien bringen `wxLC_HRULES` (horizontal) und `wxLC_VRULES` vertikal.

Als ich mit einem Auswahlwidget einen Dateibrowser schrieb, empfand ich es sehr angenehm die Pfade gleich am Label zu hinterlegen, dass nur den Dateinamen ohne Pfad anzeigte. Sie brauchten nicht parallel gespeichert werden, oder bei Aufruf aus dem Etikett zurücktransformiert werden. Da WxPerl eine Referenz am *Item* speichert, sind die Möglichkeiten groß, die Handhabung jedoch nicht die einfachste:

```
my $data = Wx::TreeItemData->new;
$data->SetData( $data_ref );
$self->SetItemData( $ItemID, $data );

Wx::Event::EVT_...( $self, -1, sub {
    my ($tree, $event) = @_;
    $tree->GetItemData(
            $event->GetItem )->GetData );
```

Dieses Beispiel war für die `TreeCtrl`, wie man an `Wx::TreeItemData` erkennt. Bei der `ListCtrl` ist es um einiges einfacher. Das Setzen wurde schon gezeigt, fürs Holen reicht `$event->GetData`. Beide Widgets kennen eine große Anzahl an Ereignissen. Das ist wichtig, da erst mit einer reichhaltigen Steuerung mit Tastatur und Maus die Benutzung des Programms Freude macht. Wie Drag'n Drop funktioniert, zeigte der letzte Teil in der vorletzten Ausgabe. Oft

geht es jedoch einfacher. Es braucht eigentlich nur Widget-ID und Item-ID in einer Paketvariable des Fensters von `EVT_LIST_BEGIN_DRAG` bis zum nächsten `EVT_LEFT_UP` hinterlegt zu werden, um auch Datensätze auch zwischen Widgets per Maus zu übertragen. Auch der `EVT_TREE_END_LABEL_EDIT` ist besonders interessant, um per `$event->Veto` unliebsame Bearbeitungen der *Item* abzuwehren.

So sehr sich `TreeCtrl` und `ListCtrl` ähneln, ersteres kennt keinen Report- und Virtuellmodus. Auch Spalten kennt nur die `TreeListCtrl` ein Hybrid aus beiden. Die `TreeCtrl` ist weniger auf Berichterstattung aus, sondern auf Interaktion und Auswahl in größeren verschachtelten Strukturen. Kein wunder also, dass es im CPAN mit dem `Wx::Perl::TreeChecker` von Simon Flack auch eine Variante gibt, in der jedes Element über eine `CheckBox` verfügt. Ein gewisser Renée Bäcker schrieb `Wx::Perl::DirTree` einen quasi fertig zu benutzbaren Dateibrowser.

```
my $tree2  = Wx::Perl::DirTree->new(
    $parent,
    [100,100], # Größe
    {
        # Wurzelverzeichnis
        dir => $path,
        # Wurzel sichtbar?
        is_root => 1,
        # nur Verzeichnisse wählbar
        allowed => wxPDT_DIR,
        # auch wxPDT_FILE möglich
    }
);
```

Aber zurück zur `TreeCtrl`. Sie kommt ohne eine Klasse `TreeItem` aus. Auf alle Eigenschaften eines *Item* können direkt die Methoden der Baumklasse zugreifen, die passende ID vorausgesetzt. Nur das Anbringen von zusätzlichen Daten ist etwas umständlicher.

```
my $ItemID = $self->AppendItem(
   $ElternID, "Label", -1, -1,
   Wx::TreeItemData->new( $data_ref )
);
```

Um Knoten (*Item*) zu erzeugen, sind `AppendItem` (neues letztes Kind des angegebenen Knotens) und `PrependItem` (neues erstes Kind des ersten Parameters) empfehlenswert. `InsertItem` arbeitet genauer, da man mit dem zweiten Parameter sagen kann, nach welchem Kind des ersten Parameters der Neue sich einschiebt. Das ist eigentlich unnötig, da per `GetItemParent` der Elternknoten bekannt ist. Passen beide Angaben nicht zusammen verhält es sich wie ein `Prepend`. Die *Item*-ID ist übrigens ein Objekt das mit der Methode `IsOk` sagen kann, ob sie auf einen Baumknoten verweist. Nach dem *Label* können allen Methoden zur Knotenerzeugung noch zwei Indizes der zugewiesenen `ImageList` erhalten. Das zweite Icon wird nur angezeigt wenn sich der Mauscursor über dem *Item* befindet.

Einmal durch den Baum zu wandern kann trotz der Mehrzahl an Methoden für den Nachbarn, Kind und Elter haarig werden. Am besten man fügt dem Baum
eine Methode zu.

```
sub DoTraverse {
    my( $tree, $parent ) = @_;
    my( $id, $cookie ) =
            $tree->GetFirstChild( $parent );
    while( $id->IsOk ) {
        if( $tree->ItemHasChildren( $id ) ) {
            $tree->DoTraverse( $id, -1 );
        } else {
            # Knoten hat keine Kinder mehr
        }
        ( $id, $cookie ) =
             $tree->GetNextChild(
                     $parent, $cookie );
    }
}
```

Es ist etwas schade, dass man hier nicht ohne "Kekse" (Lesezeichen) von einem Kind zum nächsten kommt. `GetPrevSibling` und `GetNextSibling` können das.

Falten bedeutet (meist mit einem Linksklick) alle Kinder eines Knotens unsichtbar werden zu lassen. Die so treffbare Auswahl an sichtbaren *Item* ist ein Hauptgrund für dieses Widget.

```
$tree->Collapse($ItemID);
# falte alle Kinder mit
$tree->CollapseAllChildren($ItemID);
# falte jeden knoten
$tree->CollapseAll;
```

Das Gegenteil zu jeder der drei Methoden beginnt mit `Expand`. `Toggle($ItemID)` (einzige Version) wechselt zwischen beiden Zuständen. Das Drag'n Drop innerhalb eines Baumes ist sehr einfach, da sowohl in `OnBeginDrag` (`EVT_TREE_BEGIN_DRAG`) als auch in `OnEndDrag` (`EVT_TREE_END_DRAG`) mit `$event->GetItem` bestimmen lässt, welches *Item* gezogen wurde und wo es landet.

Aussicht

Dies waren längst nicht alle Widgets. In der zweiten Hälfte dieses Rundgangs werden Texteingabefelder und das `Wx::Grid` Schwerpunkte sein.

Leserbriefe

Hallo Renée, Sebastian,

ich fand den Artikel "Perl in Multithreaded Programmen einbetten" als den besten in der aktuellen Ausgabe, gleich gefolgt von "Perl für Vortragende". In diesem Leserbrief möchte ich ein paar Gedanken äußern, die mir beim Lesen des ersten kamen. Zuvor jedoch eine kleine Anmerkung, ich habe schon einige Leserbriefe geschrieben. Jedes mal stellt sich mir die Frage, wohin ich sie schicken soll. Dann durchforste ich die Zeitschrift, finde das Impressum und stecke fest - keine Email Adresse für Zuschriften, keine WEB Seite. Ich glaube kaum, dass die Redaktion Leserbriefe an die postalische Anschrift erwartet. Klar, irgendwie findet man feedback@foo-magazin.de auf der WEB Seite, die man zuvor erraten hat. Ich wünschte mir einen einfacheren Weg.

Nun zum eigentlichen Thema. Ich kenne collectd nicht, jedoch ein wenig modperl. Soweit ich verstanden habe ist in collectd die Zuordnung von Thread zu Perl Interpreter unveränderlich, nachdem der Interpreter erzeugt wurde. Benötigt ein Thread einen Interpreter wird er erzeugt und bleibt danach mit dem Thread verbunden. Weiterhin habe ich verstanden, dass collectd recht zeitgenau regelmäßig Messungen durchführt. Das Erzeugen eines Perl Interpreters kann jedoch aufwändig sein, je nachdem wie groß der Vater ist. Verfälscht das nicht die erste Messung? Wenn das ein Problem ist, könnte man gleich beim Start ein paar Interpreter erzeugen und diese nach Bedarf zuweisen. In der Pause nach der Messung kann man den Pool dann wieder auffrischen.

Perl Interpreter können recht viel RAM verbrauchen. Modperl umgeht das Problem im prefork MPM, indem es den Kernel mittels fork() den Interpreter kopieren lässt. Dabei werden zunächst nur Referenzzähler pro Speicherseite erhöht. Der Speicher an sich wird von beiden Prozessen gleichzeitig genutzt. Erst bei schreibendem Zugriff eines der beiden Prozesse wird die Speicherseite kopiert. Bei Verwendung eines multi-threaded MPM werden Interpreter allerdings mittels perl_clone kopiert. Selbst bei geringen Interpreter-Zahlen (einstelliger Bereich) werden damit beachtliche Prozessgrößen erreicht. Ist das für collectd ein Problem? Wenn ja, gibt es irgendwelche Tricks, wie damit umgegangen wird? Möglicherweise ist das bei heutigen RAM Größen auch nicht mehr ein so großes Problem wie noch vor 10 Jahren.

Eine weitere Frage ergab sich bzgl. der Benutzung des perl_mutex in call_perl. Angenommen es sind recht viele Perl-basierte Plugins. Bei der ersten Messung werden also viele Interpreter gleichzeitig gestartet. Der gesamte Prozess ist allerdings durch den Mutex serialisiert. Eigentlich bräuchte man den Mutex nur für den kurzen Abschnitt, wenn der Interpreter in die erwähnte Liste eingefügt wird. *initial_perl sollte zum Zeitpunkt des Klonens unveränderlich sein. Es ist daher unwichtig wie viele Threads ihn gleichzeitig kopieren.

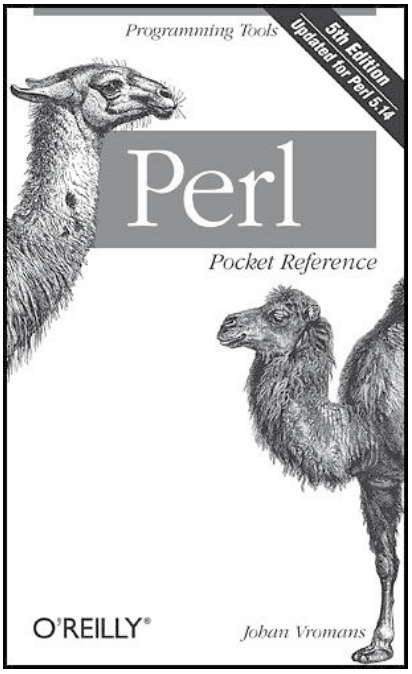
Wäre es nicht ausreichend allein das Einfügen in die Liste zu serialisieren:

```
t->interp=
      perl_clone(initial_perl, clone_flags);
pthread_mutex_lock(&perl_mutex);
insert_list(t);
pthread_mutex_unlock(&perl_mutex);
```

Man könnte das Einfügen in die Liste ggf. auch nach dem Aufruf des Plugins erledigen. Dann ist man im kritischen Bereich vom Trigger bis zur Messung lock-frei.

Torsten Förtsch

Buchverlosung



In den Rezensionen hat Herbert Breunung "Perl - Pocket Reference" aus dem O'Reilly-Verlag vorgestellt. Er hat uns ein Exemplar des Buches (in englischer Sprache) für eine Verlosung zur Verfügung gestellt. Wir verlosen deshalb dieses eine "Perl - Pocket Reference" unter allen Einsendungen, die bis zum 9. März 2012 folgende Frage beantworten:

Wofür setzen Sie Perl ein?

Bitte schicken Sie eine Mail mit der Antwort an *verlosung@perl-magazin.de*.

TPF News

Improving Perl 5

Im November hat Nicholas Clark über 144 Stunden an Verbesserungen von Perl 5 gearbeitet. Neben Bugfixes hat er hauptsächlich an der Verbesserung des *buildtoc*-Skripts gearbeitet. Für den Perl-Benutzer selbst bedeutet das keine wirkliche Veränderung, aber für die Pumpkings von Perl ist das eine Arbeitserleichterung.

Außerdem war Clark damit beschäftigt, einen Bug unter AIX zu fixen, der sich als Bug im Compiler herausgestellt hat. Clark hat einen Workaround dazu in Perl bereitgestellt. Nicholas Clark arbeitet seit über vier Monaten an Perl 5. Im November wurde der Grant erstmalig verlängert. Damit ist es für Clark möglich, weitere 400 Stunden an Perl 5 zu arbeiten.

Bezahlt wird Nicholas Clark aus dem Topf *Perl 5 Maintenance*, der Mitte des Jahres geschaffen wurde.

Google Code-In 2011: Perl Foundation und Parrot Foundation sind dabei

Beim diesjährigen Google Code-In wurden sowohl die Perl Foundation als auch die Parrot Foundation als Organisation angenommen. Der Google Code-In richtet sich an Schüler zwischen 13 und 17 Jahren. Die Perl Foundation hat einige Ideen gesammelt, die von den Schülern umgesetzt werden können.

Für eine abgeschlossene Aufgabe gibt es für die Schüler ein T-Shirt. Wenn mehrere Aufgaben abgeschlossen werden, gibt es bis zu 500 USD. Um an der Veranstaltung teilnehmen zu können, brauchen die Schüler einen Google-Account. Mehr Informationen dazu gibt es unter http://www.google-melange.com/gci/document/show/gci_program/google/gci2011/faq#claim

Makros werden in Rakudo eingeführt - Hague Grant akzeptiert

Carl Mäsaks Antrag auf einen Hague Grant wurde akzeptiert. Ziel des Grants ist es, Makros in Rakudo zu implementieren. Der Antrag ist unter http://news.perlfoundation.org/2011/09/hague-grant-application-implem.html zu finden.

Grant für Arbeiten an Test::Builder2 abgeschlossen

Nach einem sehr langen Zeitraum von rund 3 Jahren hat Michael G. Schwern den Grant für die Arbeiten an Test::Builder2 abgeschlossen. In seinem Abschlussbericht erklärt Schwern ausführlich, welche Arbeiten er in der Zeit geleistet hat. Test::Builder ist das Framework, auf dem viele Test-Module auf CPAN aufbauen, ist also ein wichtiges Werkzeug für Perl-Programmierer.

Booking.com spendet 100.000 EUR an Perl-Foundation

Booking.com hat eine weitere großzügige Spende an die Perl Foundation getätigt: Diesmal wurde ein Scheck über 100.000 EUR überreicht. Das Geld fließt in den Topf für Arbeiten am Kern von Perl 5. Aus diesem Topf werden zur Zeit Dave Mitchell und Nicholas Clark bezahlt, um Bugs zu fixen, die sonst keiner anfassen möchte und um den Kern von Perl 5 zu verbessern.

Fixing Perl 5 Core Bugs

Seit knapp zwei Jahren arbeitet Dave Mitchell sehr intensiv am Kern von Perl 5. Im November hat er hauptsächlich daran gearbeitet, die Codeausführung innerhalb von regulären Ausdrücken zu verbessern bzw. Fehler zu beheben, die damit zu tun haben.

Mit `/(?{...})/` ist es möglich, während der Ausführung von regulären Ausdrücken Perl-Code auszuführen. Dieses Feature gibt es schon seit längerem, es ist aber als experimentell gekennzeichnet. Durch die Arbeiten von Mitchell ist dieses Feature ein Stück weit näher an "stabil" herangerückt. Im Oktober wurde der Grant erneut verlängert und Geld für weitere 400 Stunden genehmigt.

CPAN News XXl

Test::Mock::Cmd

Ein weiteres Test-Modul in dieser Liste:<`Test::Mock::Cmd`. Nicht selten werden andere Programme mittels backticks, `qx//` oder `system()` aufgerufen. Aber um grundsätzlich Funktionalitäten zu testen, braucht man bestimmte Ausgaben der Programme und/oder man kann sich nicht unbedingt darauf verlassen, dass die Programme installiert sind. Für diese Fälle wurde `Test::Mock::Cmd` entwickelt.

```
use Test::Mock::Cmd \&my_cmd;

my $perl_v = qx{perl -v};
print $perl_v,"\n";

sub my_cmd {
    my $command = shift;

    if ( $command eq 'perl -v' ) {
        return "This is perl, v1.0.0";
    }
    else {
        return "Ungueltiges Kommando";
    }
}
```

App::cpackage

Es geht hier weniger um das Modul als vielmehr um das Skript *cpackage*, das mit dem Modul mitgeliefert wird. Das hilft in Situationen, in denen man Module und seine Abhängigkeiten weitergeben möchte. Allerdings nicht als ausführbare Datei, denn dafür gibt es andere Möglichkeiten wie z.B. `PAR::Packer`, sondern als Sammlung von Modulen.

`cpackage` erstellt ein Paket mit dem Modul und den Abhängigkeiten sowie ein Programm *install.pl*, mit dem dann die Pakete installiert werden können. Dadurch, dass alle notwendigen Distributionen im Paket enthalten sind, kann man die Module auch ohne Internetzugang installieren.

Beim Erstellen, des Pakets kann man verschiedene Optionen angeben. So kann man als Quelle für die Pakete auch einen eigenen CPAN-Spiegel angeben (siehe auch den extra Artikel in dieser Ausgabe).

```
cpackage My::CPAN::Module
cpackage --mirror http://cpan.local
         --mirror-only My::Private::Module
```

SVG::Calendar

Das Jahr ist noch jung. Da ist ein Kalender fällig. Mit `SVG::Calendar` kann man sich sehr einfach eigene Kalender erstellen lassen, auch mit eigenen Bildern.

```
use SVG::Calendar;

my $year = '2012';
my %options;

$options{image} = {
   map{
        my $month = sprintf "%02d", $_;
        "$year-$month" => "./$_.jpg";
    }(1..12)
};

my $svg_c = SVG::Calendar->new(
    page => 'A4',
    %options,
);

$svg_c->output_year(2012, 'FooMagazin');
```

String::Dump

Warum wird nicht das ausgegeben, was ich eigentlich erwarte? Vielleicht, weil in dem String etwas anderes drinsteckt als erwartet. In solchen Fällen muss man genauer hinschauen - vielleicht auch etwas Nicht-druckbares sichtbar machen. Hier kann dann `String::Dump` weiterhelfen. Das Modul liefert ein paar Hilfsfunktionen, um den String z.B. im Hex-Format anzuzeigen:

```
use utf8;

 {
     no utf8;
     say dumpstr('Ĝis! ☺');
     # C4 9C 69 73 21 20 E2 98 BA
 }

 say dumpstr('Ĝis! ☺');
     # 11C 69 73 21 20 263A

 my $string = 'Ĝis! ☺';
 say dumpstr( hex => $string ); # wie oben
 say dumpstr( oct => $string ); # Oktal

 say dumpstr( bin => $string ); # Binaer
```

Test::Continuous

In dieser Ausgabe wurde ein ganzer Artikel dem Thema *Continuous Integration* gewidmet. Dort wurde unter anderem auf Commits gelauscht. Eine Stufe vorher setzt `Test::Continuous` an. Hier werden die Tests bereits bei jedem Speichern ausgeführt.

Praktisch ist es, wenn man Desktop-Benachrichtigungen verwendet, so dass man wirklich sofort sieht wenn etwas nicht so läuft wie es soll.

```
cd Modulverzeichnis
autoprove -Ilib
```

Begin::Declare

Häufig werden Variablen schon während der Kompilierung gebraucht, und zwar mit Werten. Bei `my $var = 1` wird die Variable zwar beim Kompiliervorgang schon angelegt, aber nicht mit dem Wert "1" belegt. Das passiert erst zur Laufzeit. Man sieht daher in vielem Code so etwas wie

```
my $dir;
BEGIN {
    $dir = '/pfad';
}
use lib $dir;
```

Das ist nicht unbedingt hübsch, führt aber zum Ziel. Mit dem Modul `Begin::Declare` wird das übersichtlicher und das Modul erzeugt mit `MY` lexikalische Variablen. Außerdem stehen auch `OUR` und `STATE` zur Verfügung.

```
use Begin::Declare;

my $dir = '/pfad';
use MyTest $dir;

MY $path = '/pfad2';
use MyTest $path;
```

Termine

Februar 2012

02. Treffen Dresden.pm
04.-05. FOSDEM
06. Treffen Hamburg.pm
07. Treffen Stuttgart.pm
Treffen Frankfurt.pm
12. London Perl-Workshop
13. Treffen Ruhr.pom
16. Treffen Darmstadt.pm
21. Treffen Erlangen.pm
28. Treffen Bielefeld.pm
29. Treffen Berlin.pm

März 2012

01. Treffen Dresden.pm
05.-07. 14.Deutscher Perl-Workshop
05. Treffen Hamburg.pm
06. Treffen Frankfurt.pm
Treffen Vienna.pm
12. Treffen Ruhr.pm
15. Treffen Darmstadt.pm
Treffen Munich.pm
17.-18. Chemnitzer Linux-Tage
19. Treffen Erlangen.pm
27. Treffen Bielefeld.pm
28. Treffen Berlin.pm
30.-01. QA Hackathon

April 2012

02. Treffen Hamburg.pm
03. Treffen Frankfurt.pm
Treffen Vienna.pm
05. Treffen Dresden.pm
09. Treffen Ruhr.pm
16. Treffen Erlangen.pm
19. Treffen Darmstadt.pm
24. Treffen Bielefeld.pm
25. Treffen Berlin.pm
28. Grazer Linux-Tage

Hier finden Sie alle Termine rund um Perl.

Natürlich kann sich der ein oder andere Termin noch ändern. Darauf haben wir (die Redaktion) jedoch keinen Einfluss.

Uhrzeiten und weiterführende Links zu den einzelnen Veranstaltungen finden Sie unter

http://www.perlmongers.de

Kennen Sie weitere Termine, die in der nächsten Ausgabe von $foo veröffentlicht werden sollen? Dann schicken Sie bitte eine EMail an:

termine@foo-magazin.de

LINKS

http://www.perl-nachrichten.de

Unter Perl-Nachrichten.de sind deutschsprachige News rund um die Programmiersprache Perl zu finden. Jeder ist dazu eingeladen, solche Nachrichten auf der Webseite einzureichen.

http://www.perl-community.de

Perl-Community.de ist eine der größten deutschsprachigen Perl-Foren. Hier ist aber nicht nur ein Forum zu finden, sondern auch ein Wiki mit vielen Tipps und Tricks. Einige Teile der Perl-Dokumentation wurden ins Deutsche übersetzt. Auf der Linkseite von Perl-Community.de findet man viele Verweise auf nützliche Seiten.

http://www.perlmongers.de/
http://www.pm.org/

Das Online-Zuhause der Perl-Mongers. Hier findet man eine Übersicht mit allen Perlmonger-Gruppen weltweit. Außerdem kann man auch neue Gruppen gründen und bekommt Hilfe...

http://www.perl-foundation.org

Die Perl-Foundation nimmt eine führende Funktion in der Perl-Gemeinde ein: Es werden Zuwendungen für Leistungen zugunsten von Perl gegeben. So wird z.B. die Bezahlung der Perl6-Entwicklung über die Perl-Foundationen geleistet. Jedes Jahr werden Studenten beim "Google Summer of Code" betreut.

http://www.Perl.org

Auf Perl.org kann man die aktuelle Perl-Version downloaden. Ein Verzeichnis mit allen möglichen Mailinglisten, die mit Perl oder einem Modul zu tun haben, ist dort zu finden. Auch Links zu anderen Perl-bezogenen Seiten sind vorhanden.

Perl-Services.de

Programmierung - Schulung - Perl-Magazin

info@perl-services.de

BOOKING.COM

online hotel reservations

NOW HIRING!

SysAdmins

MySQL DBAs

Perl Devs

Software Devs

Web Designers

Front End Devs ...

Interested? Booking.com/jobs

Booking.com B.V., part of Priceline.com (Nasdaq:PCLN), owns and operates Booking.com (TM), one of the world's leading online hotel reservations agencies by room nights sold, attracting over 30 million unique visitors each month via the Internet from both leisure and business markets worldwide.

We use Perl, puppet, Apache, MySQL, Memcache, Git, Linux …and many more!

Established in 1996, Booking.com B.V. guarantees the best prices for any type of property, ranging from small independent hotels to a five star luxury through Booking.com. The Booking.com website is available in 41 languages and offers 120,000+ hotels in 99 countries.

- Great location in the center of Amsterdam
- Competitive Salary + Relocation Package
- International, result driven, fun & dynamic work environment