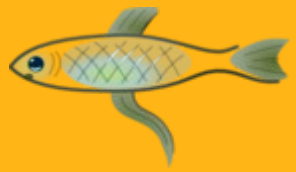
yalm magazine

07 / 2008



Fish – Friendly interactive shell

Magie der Bilder - ImageMagick

Vorwort

Arbeit geh' weg, wir kommen!

Doch vorher möchten wir euch stolz die neunte Ausgabe von Yalm präsentieren. Tatsächlich wurde das Arbeiten durch Umstände wie strahlenden Sonnenschein und azurblauen Himmel, während man sich selbst die Finger im stickigen Arbeitszimmer blutig tippte, nicht gerade erleichtert, und dennoch liegt hier die neuste Ausgabe, wieder einmal gefüllt mit lehrreichen und informativen Artikeln, vor euch.

Doch damit nicht genug. Dem aufmerksamen Leser fällt sicher auf, dass sich auch am Layout von Yalm ein wenig geändert hat – wir verwenden eine neue Schriftart, haben ein wenig Ballast über Bord geworfen und die Links deutlicher hervorgehoben. In Codezeilen wird ein layoutbedingter Zeilenumbruch jetzt mit einem kleinen Pfeil → gekennzeichnet, und umfangreiche Listings wie z.B. die des gDesklets-Artikels werden wir künftig in unserem Forum veröffentlichen, damit sie leichter zu kopieren sind.

Aber genug der einleitenden Worte. Während wir uns für die kommenden Tage eine kleine Erholungspause gönnen und uns unter wolkenlosem Himmel die Haut verbrennen hoffen wir, dass euch diese Ausgabe Spaß beim Lesen bereitet und euch mit einigen neuen Informationen versorgen kann.

Stefan Zaun
sciron@yalmagazine.org

Inhaltsverzeichnis

Rückblick

Auch im vergangen Monat gab es bedeutende Ankündigungen und anderer nennenswerte Geschehnisse, die wir euch nicht vorenthalten und hier präsentieren wollen.

»Ubuntu GNU/Linux« - Das umfassende Handbuch

Das sich nun in der dritten Auflage befindliche Werk kann nun, ganz im Sinne von OpenSource, als kostenlose HTML-Version auf der Internetpräsenz von Galileo Computing [1] heruntergeladen werden. Die mehr als 1000 Seiten umfassende Lektüre beschreibt von der Installation bis zur Administration des Systems alles Wissenswerte und richtet sich sowohl an Einsteiger, als auch an Nutzer mit fortgeschrittenen Kenntnissen.

Version 10 der OpenSource-DVD erschienen

Die Sammlung kostenloser Programme beinhaltet in der neusten Version 16 zusätzliche Applikationen und bietet somit eine Kollektion von fast 400 Programmen. Das 3,5 GB schwere ISO-Image kann gratis von der Website [2] heruntergeladen oder wahlweise im dort ebenfalls vorzufindenden Online-Shop bestellt werden.

TrueCrypt in der Version 6.0 erschienen

Das Programm, welches neben Festplatten, Teilen derselben und Wechseldatenträgern nun auch ganze Betriebssysteme verschlüsseln kann, bietet in der aktuellen Version nun auch die Möglichkeit, versteckte Container (»hidden volumes«) unter Linux anzulegen. Neben weiteren Neuerungen und zahlreicher Bugfixes ist die Leistung unter Multi-Core-Prozessoren wesentlich verbessert worden. [3]

Ubuntu 8.04.1 veröffentlicht

Am dritten Juli wurde nun das erste Point-Release für Ubuntu 8.04 LTS veröffentlicht [4], welches alle seit April 2008 erschienenen Updates beinhaltet. Die mehr als 200 Aktualisierungen beseitigen neben Sicherheitslücken auch diverse Stabilitätsprobleme. Ubuntu 8.04 Versionen, welche regelmäßigen Updates unterzogen wurden, müssen nicht neu installiert werden.

Nvidia – Keine freien Linux-Treiber

Auf Anfrage durch ZDnet ließ man verlauten, dass derzeit kein Bedarf an einem offenen Linux-Treiber bestünde. [5] Nach eigener Einschätzung stelle man hochwertige Treiber zur Verfügung, welche zum Schutz des geistigen Eigentums nicht geöffnet werden sollen.

OpenOffice.org – Erweiterung erlaubt das Editieren von PDF-Dateien

Die kostenlose Büro-Suite OpenOffice.org wird in Version 3.0 einen PDF-Editor enthalten. Die dazu benötigte Erweiterung, genannt »Suns PDF-Import-Erweiterung (SPI)«, befindet sich derzeit im Beta-Stadium und funktioniert nach Entwicklerangaben nur mit OpenOffice.org 3.0. [6] Interessenten, denen ein installierter Schnappschuss von OpenOffice.org 3.0 vorliegt, können das AddOn von der Website des Herstellers [7] beziehen.

Firefox und der Download-Weltrekord

Es ist offiziell!: Im Zuge des »Download-Days« wurde der beliebte Browser allein binnen 24 Stunden 8,002,53 Mal heruntergeladen. Damit hält man den Weltrekord für die meisten Software-Downloads innerhalb von 24 Stunden.

Stefan Zaun
sciron@yalmagazine.org

Informationen

[1] http://www.galileocomputing.de/openbook/ubuntu/
[2] http://www.opensource-dvd.de/
[3] http://www.truecrypt.org/news.php
[4] http://www.yalmagazine.org/link/38
[5] http://www.yalmagazine.org/link/39
[6] http://www.yalmagazine.org/link/40
[7] http://www.yalmagazine.org/link/41

Magie der Bilder - ImageMagick

Jeder liebt Fotos, jeder schießt Fotos und jeder möchte sie bearbeiten. Für die Bildbearbeitung unter Linux gibt es eine große Anzahl an Programmen wie z. B. Gimp, GThumb oder F-Spot. Diese Tools bieten alle eine grafische Benutzeroberfläche und mehr oder weniger Funktionalität um Bilder zu verändern. Nicht immer sind sie die erste Wahl wenn es um bestimmte Aufgaben geht. Dann wird es Zeit für den König unter den Kommandozeilen-Bildbearbeitungsprogrammen: ImageMagick.

Da staunt der Laie und der (Photoshop-)Experte wundert sich: Bildbearbeitung auf der Kommandozeile? Das kann doch nur den Linux-Geeks einfallen. Es stimmt schon - für die «08/15-Arbeiten» an Bildern und Fotos wie rote Augen entfernen, zuschneiden, skalieren usw. sind die Gimps und GThumbs dieser Welt bestens geeignet. Was aber, wenn 350 Fotos für die Publikation auf der Webpage skaliert werden müssen, damit die Oma und der Opa auch die Bilder der lieben Kleinen im Internet bewundern können?

Ein Fall für ImageMagick: Batch Conversion

Bevor der Spaß beginnt, muss ggf. über *System - Systemverwaltung - Synaptic-Paketverwaltung* das Paket **imagemagick** installiert werden. Nach der Installation sucht man in den Menüs vergebens nach einem Eintrag für ImageMagick, da es sich um ein Kommandozeilen-Werkzeug handelt. «*ImageMagick ist eine Software zum Erzeugen und Bearbeiten von Bitmap Bildern. Es kann über hundert Dateiformate lesen, schreiben und konvertieren. ImageMagick bietet 11 Grundbefehle mit über 200 Optionen zur Bildbearbeitung*». [1]

Die erste Aufgabe für ImageMagick besteht darin, eine Auswahl von Fotos zu skalieren damit sie in sinnvoller Größe via FTP auf einen Webserver kopiert werden können.

Angenommen, 100 Fotos befinden sich in einem Ordner *Bilder* und 20 davon sollen später auf der Webpage erscheinen. In einem ersten Schritt werden die 20 Fotos ausgewählt und in ein temporäres Verzeichnis kopiert. Diese Arbeit ist schnell mit dem Dateimanager erledigt. Nun kommt ImageMagick zum Zuge: Über das Menü *Anwendungen - Zubehör - Terminal* wird ein Kommandozeilenfenster geöffnet. Dort wechseln wir in das temporäre Verzeichnis (mit dem Namen *temp*) in das soeben die ausgewählten Fotos kopiert wurden. Der Befehl dazu lautet:

```
cd temp
```

Das temporäre Verzeichnis muss nicht *temp* heißen, sondern kann einen beliebigen Namen haben. Es ist wichtig festzustellen, ob man sich auch tatsächlich in diesem Verzeichnis befindet. Da wir in Kürze Löschoperationen durchführen, wäre es fatal wenn diese in einem falschen Verzeichnis ausgeführt wurden. Aus diesem Grunde sollte man mit Hilfe des Befehls

```
pwd
```

sichergehen, dass man sich derzeit tatsächlich im Verzeichnis *temp* befindet.

Wir nehmen einmal an, die Originalgrösse der Fotos beträgt 1024x768 Pixel bei Querformat oder 768x1024 bei Hochformat. Unsere Bilder sollen in das Format 640x480 (quer) bzw. 480x640 (hoch) konvertiert werden. Jetzt geht es los; folgender Befehl lässt ImageMagick die gewünschte Batch-Skalierung durchführen:

```
mogrify -monitor -resize →
640x640 *.jpg
```

Was bedeutet dieser Befehl? *mogrify* ist ein ImageMagick Kommando das Bilder in der Größe verändert und das Original ersetzt. Es ist ganz wichtig zu wissen, dass *Mogrify* die Originale im Verzeichnis löscht. Deshalb haben wir vorhin Kopien der Bilder im temporären Verzeichnis angelegt. Die unveränderten Originalfotos befinden sich nach wie vor im Ordner *Bilder*. Aus dem gleichen Grund wurde auch überprüft, ob wir uns wirklich im temporären Ordner befinden. Wem das zu heikel ist, der kann statt *Mogrify* den Befehl *Convert* verwenden. *Convert* lässt die Originaldatei im Ordner *temp* bestehen. Der Nachteil davon ist, dass man die unskalierten Bilder gar nicht benötigt und sie deshalb sowieso löschen wird.

Der Parameter *-monitor* zeigt den Fortschritt der Konvertierung für jedes einzelne Bild an. Man kann diese Option weglassen, darf sich dann aber nicht darüber wundern, dass einen Weile lang im Terminalfenster nichts passiert. Mit dem Parameter *-resize 640x640* wird ImageMagick mitgeteilt, Bilder in eine bestimmte Grösse zu konvertieren. Aber warum 640x640? Genau das ist der Clou an der Sache, die beiden Werte für die Breite und die Höhe sind nicht als ab-

solute Werte sondern als Maximalwerte zu verstehen.

Ein Foto im Querformat (z. B. 1024x768) wird damit auf die Größe 640x480 skaliert. Die x und y Werte werden von ImageMagick bei unverändertem Seitenverhältnis reduziert, bis einer der Werte die Maximalgrösse von 640 erreicht und ein anderer unter dieser Größe liegt. Derselbe Befehl auf ein Foto im Hochformat (also 768x1024) angewandt, liefert eine Skalierung auf 480x640. Das ist sehr praktisch, da man unabhängig von der Ausrichtung des Fotos nur einen Befehl braucht.

Der letzte Parameter **.jpg* ist selbsterklärend: alle Dateien, die auf jpg enden werden von ImageMagick bearbeitet. Es können natürlich auch andere Grafikformate bearbeitet werden; ImageMagick kennt sie alle. Aber Achtung, jpg ist nicht gleich JPG; Linux unterscheidet bekanntermassen zwischen Groß- und Kleinschreibung.

Wer möchte, kann den Befehl beliebig ergänzen. ImageMagick stellt hierfür eine wahre Flut an Optionen bereit. Sinnvolle Erweiterung wären z. B. die Kontrolle der JPG-Kompression. Der Standardwert ist 85 (ohne Angabe der *-quality* Option). Für eine höhere Kompression muss ein entsprechend niedriger Prozentwert (z. B. 65) gesetzt werden:

```
mogrify -monitor -resize →
640x640 *.jpg -quality 65
```

Wer sein Bild 'aufhübschen' will, der kann die Option *-equalize* verwenden. Damit unterzieht ImageMagick das Foto einer histogrammisierten Gleichmacherei; man kennt das aus anderen Grafikprogrammen wie z. B. *Gimp*. Der Befehl lautet:

```
mogrify -monitor -resize →
640x640 *.jpg -equalize
```

Wer mehr will, der findet auf der Website [2] von ImageMagick ein wahres Feuerwerk an zusätzlichen Optionen.

Nachdem unsere Bilder nun mittels *mogrify* skaliert wurden, können die Fotos via FTP oder mit einem Online-Upload auf die gewünschte Webpage hochgeladen werden.

ImageMagick zum Zweiten: Freihand Screenshot

Im zweiten Beispiel wird ImageMagick für uns eine ganz andere Aufgabe übernehmen, nämlich einen beliebigen Ausschnitt des Bildschirms fotografieren und den Screenshot anschließend auf Wunsch konvertieren.

Wer nun meint, das gäbe es schon in der Ubuntu Distribution, der hat nur zum Teil Recht. Im Menü *Zubehör* gibt es das Programm *Bildschirmfoto aufnehmen*. Damit lässt sich entweder der ganze Bildschirm oder das aktuelle Fenster fotografieren (analog zu den Tasten *Druck* und *Alt+Druck*).

Bei eingeschaltetem Compiz-Fusion hat Hardy Heron leider seine liebe Mühe mit dem Ablichten des aktuellen Fensters, es fehlt nämlich der Fensterrahmen. Eine freie Auswahl des Bildschirmauschnitts wird gar nicht angeboten. *The Gimp* macht es etwas besser; über das Menü *Datei - Holen - Bildschirmfoto* kann der ganze Bildschirm, ein beliebiges Fenster (mit Rahmen) oder ein Ausschnitt fotografiert werden. Allerdings - wer möchte jedes Mal *Gimp* hochfahren nur um einen Screenshot zu machen?

Die Lösung bringt ImageMagick: in einem Texteditor, z. B. im Menü *Anwendungen - Zubehör - Texteditor*, wird folgendes Script geschrieben und als Datei *screenshot.sh* in ein beliebiges Verzeichnis (z. B. */home/ralf/bin*) gespeichert:

```
#!/bin/bash
dateiname='screenshot.png';
import $dateiname;
display $dateiname;
```

ImageMagick Bildschirmfoto mit Bearbeitungsfunktionen

Mit der ersten Zeile wird bekannt gegeben, von welchem Kommandozeileninterpreter das Skript ausgeführt werden soll; in diesem Fall also von der *bash* Shell. In der zweiten Zeile wird eine Variable *dateiname* definiert und mit dem Wert *screenshot.png* gefüllt. Darin wird später das Bildschirmfoto gespeichert. Die dritte Zeile enthält den ImageMagick Befehl *import*. Dieser zeigt ein Fadenkreuz auf dem Bildschirm an, mit dem man einen Rahmen um einen Bildschirmausschnitt ziehen kann.

Die Handhabung ist einfach: Mit gedrückter linker Maustaste wird der Rahmen aufgezogen; beim Loslassen der Maustaste wird die Ausschnitt in die Datei *screenshot.png* gespeichert. Damit wäre die Aufgabe eigentlich erledigt. Die Datei mit dem Bildschirmfoto liegt im selben Verzeichnis, aus dem die Skriptdatei gestartet wird (dazu später mehr). Wer es etwas komfortabler mag, der kann die vierte Zeile anhängen. Der ImageMagick Befehl *display* zeigt die angegebene Datei an und stellt dazu auch noch ein Menü mit Funktionen zur Bearbeitung der Datei dar.

Das Bild auf der vorherigen Seite zeigt rechts ein Bildschirmfoto des Hardy Heron Reihers. Durch einen Mausklick in das Bild erscheint das Fenster *Commands* mit einer Vielzahl von Bearbeitungsfunktionen. Die Beschreibung der einzelnen Funktionen geht über den Umfang dieses Artikels hinaus, es lohnt sich aber die diversen Funktionen einmal auszuprobieren. Wendet man eine der Funktionen auf das Bild an, so wird die Auswirkung direkt im Bild angezeigt. Die Veränderungen sind nur von Dauer, wenn das Bild über das Menü *File - Save* gespeichert wird.

Konfiguration des Panelstarters

Wirklich bequem wird die Screenshot-Funktion erst, wenn sie schnell und einfach gestartet werden kann. Um dies zu erreichen, legen wir einen Starter im GNOME-Panel an. Dazu klickt man mit der rechten Maustaste in das Panel und wählt die Option *Zum Panel hinzufügen*. In der nun folgenden Liste ist der oberste Eintrag *Benutzerdefinierter Anwendungsstarter* die richtige Wahl. Als Typ muss *Anwendung* beibehalten werden; den Namen kann man frei vergeben. Bei *«Befehl»* muss der Pfad zur Skriptdatei angegeben bzw. ausgewählt werden. Der Kommentar ist nicht zwingend; er erscheint zusammen mit dem Namen, wenn die Maus über den Starter bewegt wird. Ein passendes Icon für den Starter kann nach einem Klick auf das Symbol links oben ausgesucht werden. Nach dem Schließen des Dialogs erscheint unser neuer Starter im Panel.

Nun noch schnell ausprobieren ob unser Eigenbau funktioniert: Nach dem Klick auf das Starter-Icon im Panel sollte das Fadenkreuz für die Auswahl des Bildschirmbereichs erscheinen. Wir ziehen einen Rahmen und erwarten das Bildschirmfoto in einem eigenen Fenster. Ein Klick in dieses Fenster sollte sich das ImageMagick Menü öffnen und nach einem weiteren Klick in das Foto wieder schließen. Die Datei müsste sich nun im Home Verzeichnis befinden (und nicht etwa im Verzeichnis in dem das Skript liegt).

Ein Tipp zum Schluss: meistens wird ein Screenshot erstellt, um einen Bildschirmausschnitt in ein anderes Dokument hinein zu kopieren. Da wäre es praktisch, den Screenshot direkt aus der Zwischenablage kopieren zu können. Leider kennt ImageMagick keinen Befehl, um eine Datei in die Zwischenablage zu übertragen bzw. den Screenshot direkt dort abzulegen. Auch der Menübefehl *Kopieren* überträgt das Foto nicht in die Gnome- oder KDE Zwischenablage, sondern in einen eigenen Zwischenspeicher, der uns nicht weiterhilft. Um das Bildschirmfoto trotzdem einfach in ein Dokument zu übertragen, kann mittels drag`n`drop die Datei aus dem Home-Verzeichnis direkt in das Dokument gezogen werden.

Ralf Hersel
rhersel@yalmagazine.org

Informationen

[1] Homepage: http://www.imagemagick.org/script/ index.php

[2] ImageMagick Options: http://www.imagemagick.org/script/command-line-options.php

GnuPG

Ist euch bewusst, dass in den *From:*-Header einer E-Mail sich alles, auch der eigene Name durch eine fremde Person schreiben lässt? Dass jeder in den Weiten des Internets auf die eine oder andere Weise eure Identität annehmen kann? Auf das erste Beispiel bezogen lässt sich das in der Regel nicht verhindern, aber man kann solche E-Mails entkräften, indem man eigene Mitteilungen konsequent signiert.

Klassische Methoden zur Verschlüsselung benutzen nur einen Schlüssel. Der Sender verschlüsselt seine Nachricht mit einem Schlüssel, der Empfänger entschlüsselt ihn mit demselben wieder. Solche Verhaltensweisen nennt man symmetrische Verfahren. Damit das jedoch funktioniert, muss der Empfänger den Schlüssel bereits besitzen und diesen vorher über einen sicheren Kommunikationskanal erhalten haben. Ansonsten könnten Unbefugte in Kenntnis des Schlüssels gelangen. Als Resultat benötigt man also einen sicheren Kommunikationskanal. Doch, sofern man einen solchen besitzt, erübrigt sich auch eine Verschlüsselung.

Public Key Verfahren (auch: asymmetrischen Verfahren) beseitigen dieses Problem, indem zwei Schlüssel erzeugt werden: Der öffentliche, welcher über beliebige Kommunikationskanäle verschickt werden kann und der private, den nur der Besitzer kennt. Idealerweise ist der private Schlüssel nicht mit dem öffentlichen rekonstruierbar. Der Sender verschlüsselt die Nachricht mit dem öffentlichen Schlüssel des Empfängers. Entschlüsselt wird die Nachricht dann mit dem privaten Schlüssel des Empfängers. Nach diesem Schema kann man demnach effektiv verschlüsseln, ohne über einen sicheren Kommunikationskanal zu verfügen. [1]

Digitale Unterschriften hingegen sollen die Authentizität einer Nachricht beweisen. Würden Nachrichten von offizieller Seite signiert, wäre es deutlich schwerer, mit gefälschten Nachrichten Unruhe oder Schaden anzurichten. Eine digitale Signatur wird mit Hilfe des privaten Schlüssels aus dem Text erzeugt. Diese kann im Anschluss daran vom Empfänger mit dem öffentlichen Schlüssel des Senders überprüft werden. Dabei wird nicht nur der Absender – der einzige «Kenner» des privaten Schlüssels - überprüft, sondern auch, ob der Text unverändert angekommen ist.

Das Logo von GnuPG

Eine Schwachstelle der Public Key Algorithmen ist die Verbreitung der öffentlichen Schlüssel. Ein Benutzer könnte einen öffentlichen Schlüssel mit falscher User-ID in Umlauf bringen. Wenn dann mit diesem Schlüssel Nachrichten kodiert werden, kann der Eindringling die Nachrichten dekodieren und lesen. Wenn er sie dann noch mit einem echten öffentlichen Schlüssel kodiert an den eigentlichen Empfänger weiterleitet, fällt dieser Angriff nicht einmal auf. In der Literatur nennt man solche Angriffe *«man-in-the-middle attacks»*. Sie stellen auch bei vielen anderen Protokollen eine Bedrohung dar.

Die von GnuPG gewählte Lösung besteht im Unterschreiben von Schlüsseln. Ein öffentlicher Schlüssel kann von anderen Leuten unterschrieben werden. Diese Unterschrift bestätigt, dass der Schlüssel zu der in der UID angegebenen Person gehört. Der Benutzer kann festlegen, welchen Unterschriften er wie weit vertraut. Vertrauen ist dabei zwar reflexiv, aber nicht symmetrisch und transitiv. Ein Schlüssel gilt als vertrauenswürdig, wenn er von Leuten unterzeichnet wurde, denen man vertraut. Wenn man selbst einen Schlüssel unterzeichnet, sollte man sich sicher sein, dass man die Identität desjenigen, dessen Schlüssel man unterschreibt, kennt. Eine Möglichkeit ist es, den Schlüssel persönlich bekommen zu haben, eine andere, den Fingerprint über zuverlässige Kanäle zu vergleichen.

Wissenswertes

Der *«GNU Privacy Guard»* (GnuPG, auch bekannt als «GPG») ist ein Kryptografiesystem, welches der Ver- und Entschlüsselung, sowie der digitalen Signatur von Daten dient und auf diese Weise Integrität und Authentizität derselben gewährleistet. Als vollständige und vor allem freie Implementierung des *«OpenPGP-Standards»* (definiert im *«RFC4880»* [2]) verwendet GnuPG nur patentfreie Algorithmen und wird als freie Software unter der GNU-GPL vertrieben. Neben Linux unterstützt GnuPG ebenfalls Microsoft Windows

sowie Mac OS X als operierende Betriebssysteme.

Dieser erste Teil befasst sich neben einer Einführung zur Thematik mit der Installation des Programms selbst, sowie der Erstellung und grundlegenden Verwaltung von digitalen Schlüsseln.

Installation

GnuPG sollte problemlos über die Paketquellen installiert werden können. Die Paketbezeichnungen können – je nach Distribution – variieren, meist nennt sich das Paket jedoch ***gnupg*** oder ***gpg***.

Eine andere Bezugsquelle stellt hier die offizielle Homepage [3] dar, auf welcher neben unterschiedlichen Spiegelservern auch der Quellcode sowie diverse Endverbraucherdokumentationen angeboten werden.

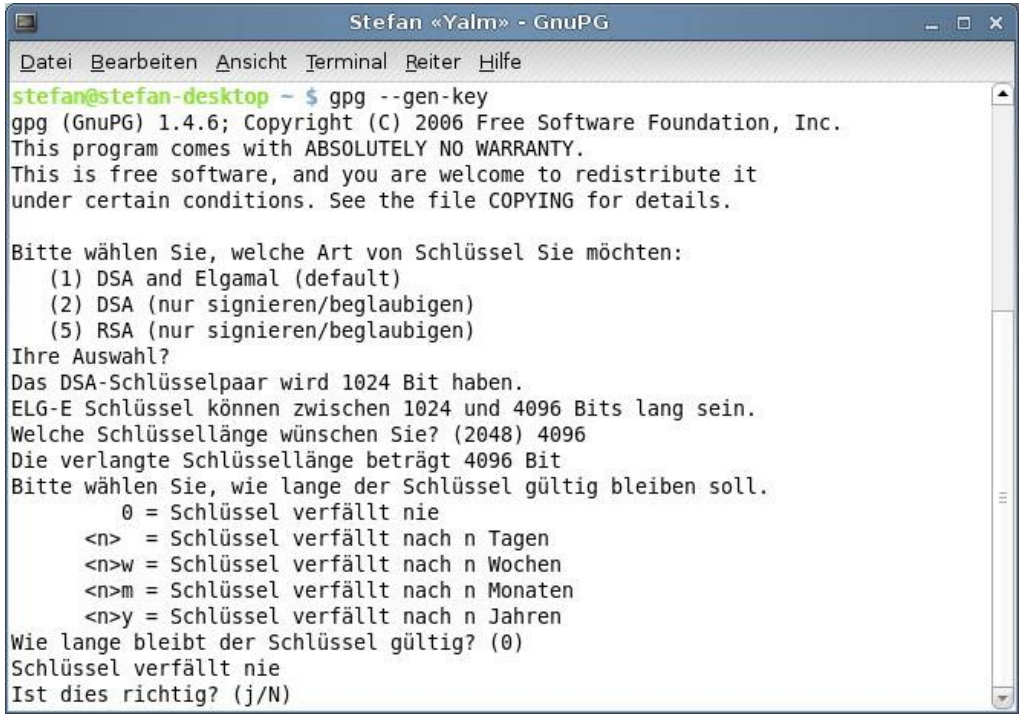
Erzeugen eines Schlüssels

Sofern man es nicht beim Verifizieren von Signaturen belassen möchte, ist ein eigener Schlüssel für die Arbeit mit GnuPG fast unerlässlich. Aus diesem Grunde erzeugen wir mit Hilfe des Terminals und dem Befehl

```
gpg --gen-key
```

zunächst ein neues Schlüsselpaar. Nun wird man nach dem zu verwendenden Algorithmus gefragt. Im Zweifelsfall sollte man sich hier, wie auch bei allen kommenden Fragen, für die Standardauswahl entscheiden, wenngleich genauere Informationen zu den einzelnen Algorithmen beispielsweise der «PGP DH vs. RSA FAQ» [4] entnommen werden können.

Für die Antwort auf die darauf folgende Frage sollte man einen Kompromiss zwischen Sicherheit und Rechenzeit finden. Je länger ein Schlüssel, desto sicherer ist dieser; aber Operationen mit bereits genanntem Schlüssel benötigen stellenweise bedeutend mehr Zeit als dies mit einem kürzeren Äquivalent der Fall wäre. Der Standard für DSA beträgt 1024 Bits, wenngleich wir, mit Hinblick auf die stetig wachsende Rechenleistung und der Tatsache, dass der Schlüssel gegebenenfalls auch noch in weiterer Zukunft genutzt werden soll, 2048 Bits empfehlen würden.



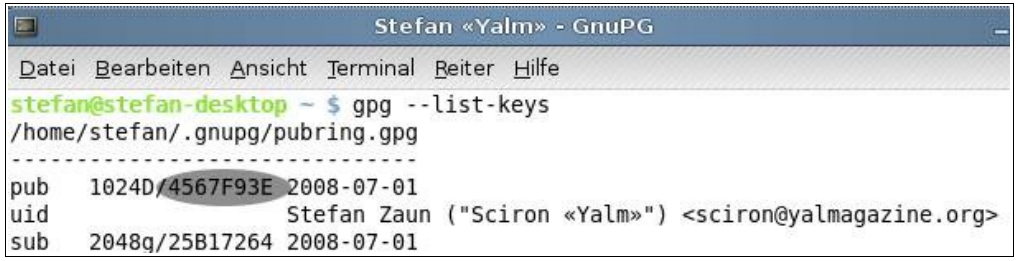
Die ersten Schritte während der Erstellung eines eigenen Schlüssels

Im Anschluss daran ist der Realname, die Email-Adresse sowie auf Wunsch ein Kommentar einzugeben. Aufgrund des Umstands, dass der Schlüssel nach dessen Generierung einer realen Person zugeordnet werden soll, sind diese Angaben zwingend und vor allem wahrheitsgetreu erforderlich. Einzelne Teile können im späteren Verlauf zwar geändert werden, es ist allerdings zu beachten, dass mit der Signatur die komplette Benutzerkennung unterschrieben wird. Wird folglich etwas geändert, gilt die Unterschrift unter den veränderten Angaben nicht länger.

Nun ist die doppelte Eingabe einer Passphrase zum Abschluss der Erstellung erforderlich. Wohlgemerkt wird nach einer Phrase, also einem Mantra, verlangt, welches bewusst länger als ein «normales» Passwort sein und weiterhin Leer- und Sonderzeichen enthalten soll. Ohne dieses Mantra ist der Schlüssel wertlos. Von daher kann es sich als sinnvoll erweisen, ein Rückrufzertifikat zu erstellen, dazu aber im Verlaufe mehr.

Das Auflisten der Schlüssel

Die «**U**ser **ID**entification number» («UID») ist eine eindeutige Buchstaben- und Zahlenfolge. Jedem Schlüssel wird hier eine Schlüssel-ID zuge-



Ein Ausschnitt der Auflistung der einzelnen Schlüssel.

wiesen, anhand derer er unmissverständlich identifiziert werden kann. Mit dem Befehl

```
gpg –list-keys
```

erhält man eine Auflistung über alle öffentlichen Schlüssel. Anhand der unten stehenden Abbildung wollen wir euch das nun Erscheinende näher erläutern:

«pub», die Kurzform von «public», kennzeichnet alle öffentlichen, das in diesem Beispiel nicht vorhandene «sec» («secret») hingegen alle «geheimen» Schlüssel. 4567F93E wäre demnach die ID des öffentlichen Schlüssels und 2008-07-01 das ISO-Datum in der Form *«JJJJ-MM-TT»* . *1024* kennzeichnet die Länge des Schlüssels von hier 1024 Bit..

Dahinter findet sich jeweils das Verfahren: ein *D* bedeutet DSA (Digital Signature Algorithm), *g* bezeichnet ElGamal. Schließlich gibt die mit *«uid»* gezeichnete Zeile die persönlichen Angaben wie den Namen oder die Email-Adresse preis.

Im- und Export von Schlüsseln

Der Export eines Schlüssels geschieht durch den Befehl

```
gpg --export [UID]
```

Ein Schlüssel wird stets mit der UID exportiert. Ohne die Angabe einer UID wird der Schlüsselbund als Ganzes abgesetzt. Die Voreinstellung bei der Ausgabe ist auf stdout gesetzt, allerdings ist es mit der Option *-o [Datei]* möglich, den Schlüssel als Datei zu speichern. Personen, denen Probleme mit dieser Vorgehensweise begegnen, sollten zusätzlich die Option *-a* verwenden. Mit diesem Parameter werden die Schlüssel nicht im Binärformat ausgegeben, sondern als ASCII (7 Bit) Dateien.

Selbstverständlich ist neben der Datei selbst auch der zu verwendende Pfad anzugeben. Wollte man nun oben aufgezeigten Schlüssel auf dem Desktop in der Datei *«szaun»* speichern, wäre dies folglich mit dem Befehl

```
gpg -o ~/Desktop/szaun.asc →
--export -a 4567F93E
```

ohne Weiteres möglich. Alternativ genügt auch an Stelle der Schlüssel-ID eine unter *«uid»* in Erfahrung gebrachte Information. Uns weiter an diesem Beispiel orientierend, hätte man mit

```
gpg -o ~/Desktop/szaun.asc →
--export
sciron@yalmagazine.org
```

das gleiche Ergebnis erzielt. Der Import funktioniert im Wesentlichen auf die selbe Weise:

Die Anweisung

```
gpg --import [Datei]
```

sollte hier vollkommen genügen.

Wenn man also nun die soeben exportierte Datei wieder importieren wollte, geschehe dies durch den Befehl

```
gpg –import ~/Desktop/szaun.→
asc
```

Nachwort

Leider zwingt uns einmal mehr die Länge zu einer Artikelreihe. Ihr kennt nun den Hintergrund und Sinn des Projekts und seit ferner in der Lage, euren eigenen und öffentlichen Schlüssel zu erstellen und zu exportieren.

In der nächsten Ausgabe wenden wir uns, jetzt, da die grundlegenden Handhabung bekannt ist, den eigentlichen Kernelementen zu: So erklären wir euch, wie ihr signiert und Signaturen prüft. Auch erfahrt ihr, auf was unter einem *Keyserver* zu verstehen und auf welche Weise dieser zu gebrauchen ist. Und ihr erlernt, wie ihr euren Schlüsselbund fortgeschritten verwalten, ein Widerruf-Zertifikat erstellen und einzelne Dateien direkt ver- und wieder entschlüsseln könnt.

Stefan Zaun
sciron@yalmagazine.org

Informationen

[1] Quelle: http://www.gnupg.org/documentation/howtos.de.html
[2] http://www.ietf.org/rfc/rfc4880.txt Erläuterungen zur RFC4880
[3] http://www.gnupg.org/index.de.html GnuPG-Homepage
[4] Auflistung der Algorithmen http://www.scramdisk.clara.net/pgpfaq.html

Fish – Friendly interactive shell

Mit hilfreichen Funktionen und einer modernen Oberfläche versucht die «friendly interactive shell» der Bash Konkurrenz zu machen. Funktionen wie Syntaxhervorhebung, eine ausgereifte und dynamische Vervollständigung oder «Multiline editing» sind sicherlich nützliche Eigenschaften. Ob sie eine ernsthafte Alternative sein kann, muss sich zeigen.

Zwar wird nahezu jedes Linuxsystem mit der Bash als Standard-Shell ausgeliefert, doch ist sie bei weitem nicht die Einzige. Eine Vielzahl mehr oder weniger bekannter Kommandozeilen steht in der Paketverwaltung bereit. Darunter alte Veteranen wie die *csh* oder die *ksh*, welche in der heutigen Zeit allerdings sehr an Verbreitung eingebüßt haben und keine ernsthaften Alternativen mehr zur Bash darstellen. Doch interessante Neuentwicklungen versuchen mit frischen Ideen und vorbildlicher Benutzerführung Anteile zu erobern.

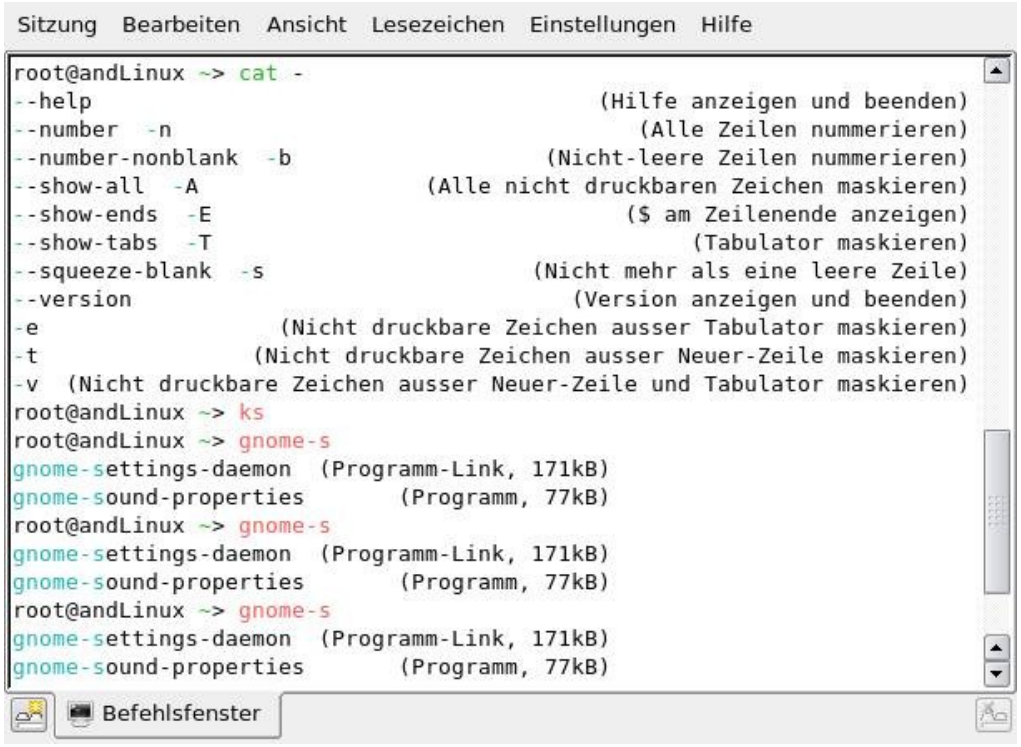
Minimalistischer Ansatz

Die von Axel Liljencrantz [1] ins Leben gerufene *«friendly interactive shell»* - Fish – versucht vieles besser zu machen und wirft einige Altlasten über Bord. Die Entwickler haben sich hohe Ziele gesteckt [2]. Um den Quellcode sauber und übersichtlich zu halten, werden so wenige Kommandos wie möglich in die Shell integriert. Was ist damit gemeint? Die Bash-Programmierer haben oft genutzte Kommandozeilenprogramme direkt in die Shell eingefügt. Sie funktionieren immer solange eine Bash verfügbar ist. Externe Programme sind hierfür nicht nötig. So verfügt die Bash beispielweise über eigene Implementierungen der Ausgabebefehle *«echo»* und *«printf»*, obwohl diese in jeder normalen Linuxdistribution mitgeliefert werden. Die Fish bringt wesentlich weniger eingebaute Befehle, sogenannte *«builtins»*, mit und verlässt sich an dieser Stelle auf die bewährten, von Linux bereitgestellten Programme. Da sich die Bash ohnehin sehr stark an den ursprünglichen Befehlen orientiert, können Kommandos wie *«echo»* auf gewohnte Art weiter genutzt werden.

Die Programmierer setzen diesen Grundsatz durchaus konsequent um. So ist die Fish auch nicht in der Lage selbstständig zu rechnen. Hier sollen stattdessen auf externe Programme wie *«bc»* und *«expr»* zurückgegriffen werden. Diese Ansätze klingen nun ziemlich minimalistisch und wenig benutzerfreundlich. Doch durch den Verzicht auf viele solcher Kleinigkeiten können sich die Entwickler genau auf die primäre Aufgabe einer Shell konzentrieren: Die Eingabe von Kommandos. Hier bietet die Fish eine Fülle von neuen Möglichkeiten und Verbesserungen gegenüber anderen Shells.

Syntaxhervorhebung

Schon bei der Eingabe der ersten Zeichen fällt eine Sache auf: Alle Kommandos werden farbig hinterlegt. Wenn Fish der Meinung ist, ein Befehl sei korrekt, wird aus der anfänglich roten Farbe ein grüner Schriftzug, der signalisieren soll, dass es sich um einen gültigen Aufruf handelt. Diese Syntaxhervorhebung ist auch in ähnlicher Form aus Editoren wie *kate*, *gnome-edit* oder *vim* be-



Hilfe und Parametervervollständigung

kannt und erlaubt das schnelle aufspüren von Buchstabendrehern oder falschen Anweisungen. Allerdings hat die Syntaxhervorhebung in Fish auch ihre Grenzen: So werden keine Parameter beim Aufruf von Programmen überprüft und auch die korrekte Schreibweise von Variablen kann nicht angezeigt werden.

Praktische Helfer

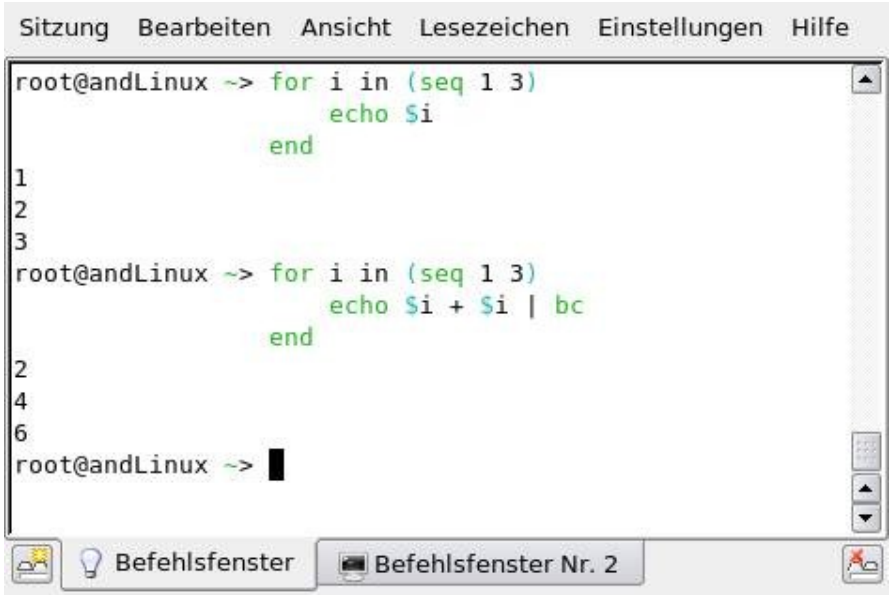
Selbstverständlich unterstützt Fish auch die von anderen Shell bekannte *«Tab completion»*, welche mit Hilfe der Tabulatortaste eine Anweisung vervollständigen kann. Während die Bash hier nur normale Aufrufe wie *«echo»* oder *«grep»* ergänzen kann, kennt die Fish zu fast allen Standard-Programmen auch die meisten Optionen und Parameter. So ist es möglich eine Menge Tipparbeit zu vermeiden. Beispielweise vervollständigt Fish den Befehl *«chown –dereference»* komplett inklusive dem Parameter.

```
cho TAB –de TAB
```

Die Fish benötigt hier nur 9 Anschläge, wohingegen die Bash mit 17 fast das Doppelte zu verzeichnen hat. Natürlich sei erwähnt, dass den meisten längeren Optionen auch ein kürzerer Alias zugeordnet ist und spätestens wenn Fish ein Programm nicht kennt, ist der Vorteil dahin. Dennoch gewöhnt man sich schnell an dieses Feature und vermisst es in anderen Shells gelegentlich. Die Eingabeunterstützung geht noch einen Schritt weiter. Fish bietet eine integrierte Hilfe zu fast allen häufig genutzten Kommandozeilenprogrammen. Oft kommt es vor, dass man nicht mehr weiß welche Option nun die gewünschte Verhaltensweise bewirkt oder wie ein bestimmter Hostname lautete. Bei all diesen Problemen kann die Shell assistieren.

```
ls –
```

gefolgt von zweimal Tabulator, lässt eine Hilfeseite erscheinen, die alle passenden Parameter mit einer kurzen Funktionsbeschreibung auflistet, ohne den eigentlichen Eingabemodus zu verlassen. So kann, ohne manpage öffnen und schließen, direkt weitergearbeitet werden. Wie oben erwähnt, können auch Hostnamen vervollständigt werden. Das ist besonders bei *«ssh»* oder auch «ping» nützlich. Fish zieht mehrere Quellen für Hostnamen heran. Einmal die *«known_hosts»* Datei von ssh im jeweiligen Heimatverzeichnis des Benutzers, die /etc/fstab für nfs Server und natürlich die /etc/hosts.



Multiline editing und Syntaxhervorherbung

Tastenkombinationen

Praktische Tastenkürzel gibt es auch einige:

Mit *«alt+w»* kann eine kurze Beschreibung des Befehls unterhalb des Cursors abgerufen werden. *«alt+l»* listet den Inhalt des aktuellen Verzeichnisses auf und «alt+p» sorgt dafür das die Ausgabe Seitenweise erfolgt [4]. Auch die Befehlshistory ist einfach zugänglich. Es genügt, die bekannten Elemente eines Kommandos einzugeben, um an den letzten passenden Befehl zu gelangen. Mit den Pfeiltasten *«Nach-Oben»* und *«Nach-Unten»* kann durch die Ergebnisse geblättert werden.

Multiline Editing

Fish unterstützt *«Multiline editing»*. Dies erlaubt das übersichtliche Strukturieren von komplizierten und langen Anweisungen (z. B. Schleifen) über mehrere Zeilen direkt in der Shell, ohne den Umweg über eine Datei nehmen zu müssen. Mit *«Alt+Return»* beginnt Fish eine neue Zeile ohne die vorherige zu verarbeiten. In der Bash ist hier mühsames separieren mit Semikola nötig um Befehle in der gleichen Zeile vonein-

ander zu trennen. Sind die Anweisungen komplett, wird mit einem abschließenden *«Return»* das gesamte Konstrukt ausgeführt.

Das Öffnen einer Datei über die Shell erfordert immer die Eingabe eines Programms, das die Datei fehlerfrei darstellen kann. Hierfür bringt Fish das Kommando *«open»* mit. Damit können Dateien mit den Standardprogrammen der Desktopumgebung geöffnet werden, z. B. eine *.odt* Datei mit OpenOffice.org Writer.

Nachteile

Nun zu einem negativen Punkt: Die Syntax der Fish ist fast vollständig inkompatibel zur Bash und zu anderen Shells. Da der weit überwiegende Teil aller Shellscripte in Bash/ Korn-Shell Syntax vorliegt, ist der Einsatz von Fish auf Servern fast ausgeschlossen. Das soll nicht heißen, dass keine umfangreichen Scripte erstellt werden können. Die Fish bietet ebenso wie viele andere Shells leistungsstarke Funktionen zum Automatisieren von Abläufen und zum Erstellen von Shellscripts. Doch gerade für Bash-Benutzer ist die neue Syntax gewöhnungsbedürftig.

Eine *For*-Schleife in Fish:

```
for i in (seq 1 3)
echo $i
end
```

Und eine in Bash:

```
for i in {1..3}; do echo $i; →
done
```

An diesem Beispiel zeigt sich auch der minimalistische Ansatz der Fish, was eingebaute Befehle betrifft. So ist mit *«seq»* ein externes Programm erforderlich um die Zählschritte zu generieren. Die Bash beherrscht das von Haus aus.

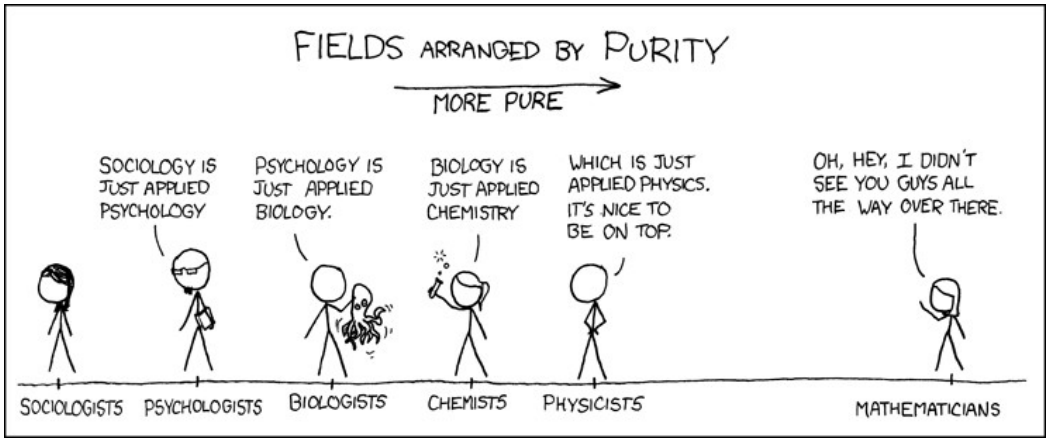
Fazit

Im Großen und Ganzen ist die Fish besonders für Shellanfänger eine super Sache und auch im Profi- und Fortgeschrittenenlager wird die freundliche Shell bestimmt einige Nutzer für sich gewinnen können. Außerdem lässt sie sich auch wunderbar und ohne jegliche Probleme neben der Bash betreiben. So steht dem Ausprobieren nichts im Wege. Wer nach ausgiebigen Tests vielleicht zu dem Schluss kommt, dass für ihn die Fish ein geeigneter Ersatz ist, kann sie mit dem Programm *«chsh»* zur Login-Shell machen.

```
chsh –s /usr/bin/fish
```

...und umgekehrt um wieder an die Bash als Login-Shell zu kommen:

```
chsh –s /bin/bash
```



Fachrichtungen nach «Ursprünglichkeit»: Soziologie – Psychologie – Biologie – Chemie... Mathematik
http://xkcd.com/435/

Maximilian Schnur
max@yalmagazine.org

Informationen

[1] Offizielle Webpräsenz
http://www.fishshell.org/

[2] Endverbraucherdokumentation
http://www.fishshell.org/ user_doc/ design.html

Software für Ubuntu

Paketverwaltung, Repository, Synaptic, Adept & Co. - Böhmische Dörfer? Dieser Beitrag taucht in die Tiefe des Themas Software ein und erklärt die Hintergründe.

Wer sich als Windows-Nutzer mit Linux beschäftigt erkennt schnell, dass sich die Art der Softwareinstallation bei den beiden Betriebssystemen unterscheidet. Zunächst möchten wir aber mit dem alten Märchen aufräumen, dass man die Programme, die man nutzen möchte, unter Linux selbst kompilieren, das heißt mit einem Compiler in Maschinensprache übersetzen muss. Für Ubuntu und seine Ableger, wie z. B. Kubuntu, stehen heute über 23.000 fertige Programmpakete zur Verfügung.

Dass von Paketen gesprochen wird hat den Hintergrund, dass nicht nur das eigentliche Programm, sondern auch alle zugehörigen Bibliotheken (z. B. Sprachdateien, Zeichensätze) in einem solchen Paket enthalten sind. Wer schon Software unter Windows installiert hat, weiß, dass man schnell mit einigen Mausklicks zum Ziel kommt. Hier werden alle Programme separat nebeneinander installiert und müssen gegebenenfalls auch einzeln per Update auf den neusten Stand gebracht werden. Auch kann es hier passieren, dass, wenn ein Programm deinstalliert wird, Bibliotheken gelöscht werden, die auch noch für andere Programme notwendig sind und diese Programme anschließend nicht mehr funktionieren.

Bei vielen Linux-Distributionen (wie Ubuntu) geht man dieses Thema mit anderen Lösungsansätzen an. Hier muss sich der Nutzer nicht um Abhängigkeiten und Unverträglichkeiten von Programmen kümmern; das übernimmt die sogenannte Paketverwaltung für ihn. Bei Ubuntu ist dies ***Synaptic*** [1], bei Kubuntu ***Adept*** [2]. Die System-Updates können hier beispielsweise frei konfiguriert werden. So kann man wählen, wie oft nach verfügbaren Updates gesucht wird, ob man informiert werden möchte, so dass man selbst entscheidet welche Programme auf den neuesten Stand gebracht werden sollen oder ob Sicherheits-Updates oder sämtliche Updates automatisch im Hintergrund ablaufen sollen. Sogar den gewünschten Server kann man auswählen. Ein lästiger Neustart ist meistens nicht nötig. Alle Linux-Updates stehen sowohl für das Betriebssystem selbst wie auch für die per Paketverwaltung installierten Programm zur Verfügung.

Nutzer- und softwarespezifische Daten werden unter Windows in einer Registrierdatenbank, der Registry, gespeichert. Je mehr Programme installiert werden, desto langsamer wird der Windowsrechner. Selbst nach der Deinstallation eines Programms verbleiben häufig Einträge in der Registry, die zu Performance-Einbußen führen können.

Bei Linux-Betriebssystemen werden für jedes installierte Programm Textdateien im Verzeichnis */etc* oder */usr/local* installiert, die von einem kundigen Benutzer auch selbst editiert werden können. Auf diese so genannten Scripte greift die Software zu, um die für einen sicheren Betrieb notwendigen Daten zu finden. Die schon oben erwähnten Bibliotheken werden im Verzeichnis bzw. Unterverzeichnis */lib* abgelegt und stehen den von ihnen abhängigen Programmen zentral zur Verfügung. Der Vorteil hierbei ist, dass die Geschwindigkeit des Rechners nicht von der Anzahl der installierten Programme beeinträchtigt wird. Bei Ubuntu & Co ist somit immer zügiges Arbeiten gewährleistet.

Repositories

Bei den Repositories [3] handelt es sich um Bezugsquellen, die in die Bereiche *Main*, *Restricted*, *Universe* und *Multiverse* unterteilt sind. Sie stellen ihre Pakete im Internet zur Verfügung, wo sie auf Servern abgelegt sind und zum Download bereitstehen.

Im *Main-Repository* befinden sich Programme, die von den Ubuntu-Entwicklern betreut werden und den Ubuntu-Lizenzanforderungen entsprechen. Wie der Name schon sagt, sind hier die fundamentalen Programme für den Betrieb von und für die Arbeit mit Ubuntu zu finden.

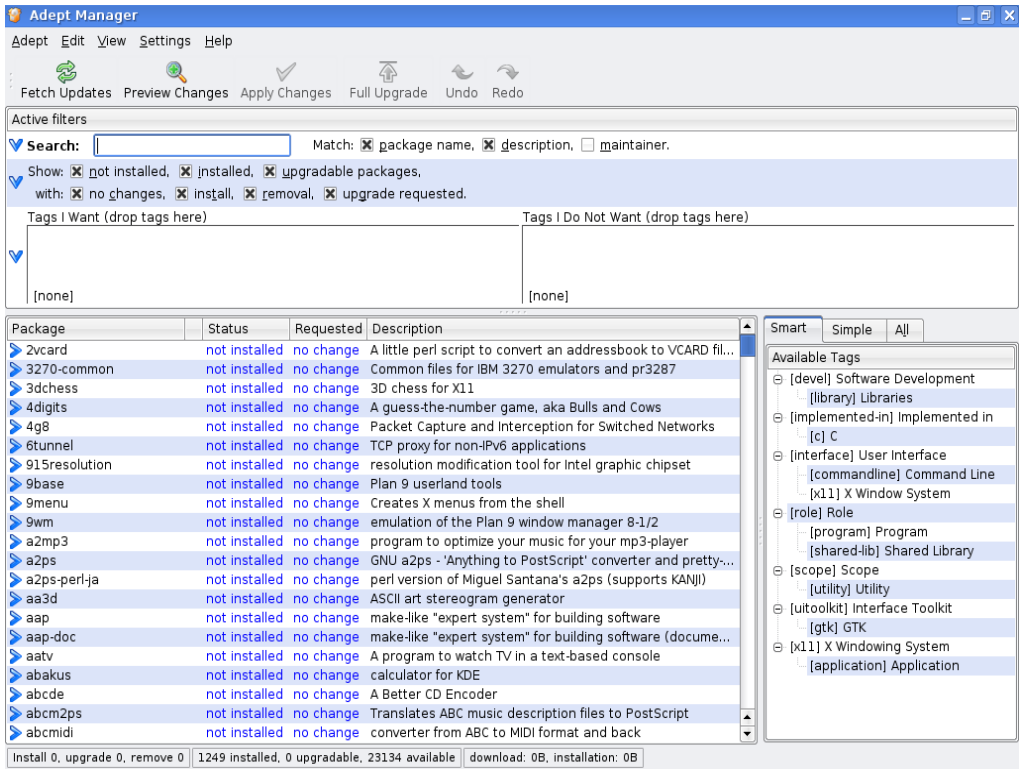
Für den Bereich *Restricted* gilt das nur eingeschränkt, da sich hier Software befindet, auf deren Quellcode auch die Ubuntu-Entwickler keinen Zugriff haben, wie beispielsweise externe Hardware-Treiber.

Universe und *Multiverse* werden vom Ubuntu-Team nicht supportet, es werden keine Bugfixes durchgeführt. Hier finden wir häufig proprietäre Software, die als *deb*-Paket vorliegt. Der Einsatz dieser Programme liegt in der Verantwortung des Nutzers.

Die Paketverwaltung [4]

Bei diesem Thema gehen die verschiedenen Linux-Distributionen unterschiedliche Wege. Voraussetzung zur Installation von Software ist neben einem CD/DVD-Laufwerk eine bestehende Internetverbindung.

Da sich die Ubuntu- und Kubuntu-Lösungen sehr ähneln, greifen wir einfach die KDE-Variante Adept *(s. Abb.)* heraus und zeigen wie einfach man mit ihr arbeitet. Wir finden das Programm unter *System – Adept* und starten es per Mausklick. Nachdem wir unser Administratoren-Passwort eingegeben haben, startet Adept. Möchten wir nun ein bestimmtes Paket installieren, so geben wir dessen Namen in die Suchzeile ein. Adept findet nun alle Pakete die diesen Suchbegriff enthalten und zeigt deren Status an. Das gewünschte Paket wird angeklickt, und es öffnet sich ein Fenster, das uns weitere Informationen liefert. So werden Installationsumfang, Versionsnummer und eine Beschreibung des Programms angezeigt. Sind wir einverstanden, bestätigen wir mit Installation anfordern und anschließend Änderungen anwenden. Nun wird das Paket heruntergeladen, Abhängigkeiten werden überprüft und es wird installiert.



Adept ist sehr übersichtlich und einfach zu bedienen

Die Konfiguration

Oben links in der Menüzeile finden wir den Eintrag *Adept*.. Wir klicken darauf und in dem sich öffnenden Fenster auf Aktualisierungen holen. Jetzt werden in all den freigegebenen Softwarequellen nach Aktualisierungen gesucht und möglicherweise zur Installation vorgeschlagen.

Wir klicken *Adept-Paketquellen* verwalten an und können das gewünschte Repository ankreuzen und den Downloadserver festlegen. Im Bereich *Third-Party-Software* kann man einfach gewünschte Bezugsquellen, die hier nicht aufgeführt wurden, eintragen. Hierzu klicken wir auf den Button Hinzufügen und schreiben oder besser kopieren die Bezugsadresse in die vorgesehene Zeile und bestätigen mit *OK*. Zukünftig können wir auch aus diesem Repository Software per Adept installieren. Im Bereich *Updates* legen wir den von uns gewünschten Modus fest und beenden die Aktion mit Schließen.

Der GDebi-Installer

Zuweilen laden wir aus dem Internet Software herunter und finden sie anschließend auf unserem Desktop wieder. Durch einen Mausklick mit der rechten Taste auf das Icon öffnet sich ein Fenster, in dem wir die Option «*öffnen mit - GDebi Package Installer*» auswählen. Nun sehen wir eine kurze Beschreibung des Programms und können uns Details und die enthaltenen Dateien nebst vorgesehenem Speicherort anschauen. Sind wir einverstanden, klicken wir auf Paket installieren und der Installer beginnt mit seiner Arbeit. Sobald er fertig ist, bekommen wir es mitgeteilt und können, nachdem wir GDebi verlassen haben, sofort mit dem installierten Programm arbeiten.

Installation per Terminal

Linux wäre nicht Linux, wenn wir nicht auch auf die grafischen Helfer verzichten und einfach per Kommando in einem Terminal Software installieren könnten. Das hierfür notwendige Werkzeug heißt *Advanced-Packaging-Tool* oder kurz APT [5]. Es pflegt unser System, sucht nach Programmen, installiert und deinstalliert sie

[6]. Die Programmquellen finden wir in */etc/apt/sources.list*, die wir per Editor, unter KDE z. B. Kate, bearbeiten können. Per Befehl

```
sudo kate →
/etc/apt/sources.list
```

öffnen wir den Editor und finden für gewöhnlich die oben genannten Repositories aufgeführt.

Vor der Bearbeitung dieses Scripts machen wir zur Sicherheit eine Kopie, die wir im Notfall wieder einspielen können. Sollten wir vor einer gewünschten Bezugsquelle das #-Zeichen finden, so müssen wir es entfernen. Das Zeichen dient dazu, in Scripten etwas zu kommentieren, die etwaigen nachfolgenden Befehle werden nicht ausgeführt. Nach der Entfernung der #-Zeichen ist das Script zu speichern und schon steht uns die neue Quelle zur Verfügung.

Eine typische Zeile in der sources.list sieht beispielsweise so aus:

```
deb →
http://security.ubuntu.com/
ubuntu gutsy-security main
restricted
```

Um die freigegebenen Quellen nutzen zu können geben wir in einem Terminal den Befehl

```
sudo apt-get update
```

ein und das repository wird erfasst. Um nun Software zu installieren lautet der Befehl

```
sudo apt-get install Paketname
```

Dabei müssen wir nicht ein Paket nach dem anderen installieren, sondern können, wie hier zu sehen, durch Leerzeichen getrennt, auch beliebig viele Namen hintereinander eingeben:

```
sudo apt-get install →
Paketname1 [Paketname2] →
[Paketname3].
```

Um Software wieder von unserer Maschine zu verbannen nutzen wir

```
sudo apt-get remove Paketname
```

Falls auch alle Einstellungen gelöscht werden sollten nehmen wir diesen Befehl:

```
sudo apt-get purge Paketname
```

Unser CD- oder DVD-Laufwerk binden wir als Lieferant von Software ein:

```
sudo apt-cdrom add
```

Programme kompilieren

Ist uns das gewünschte Programm nur im Quellcode zugänglich, so müssen wir es kompilieren. Profis nutzen diesen Weg auch gerne, um Software perfekt an die vorhandene Hardware anzupassen. Die vorgefertigten *deb*-Pakete sind hier ein Kompromiss und nicht immer für das eigene System optimal. Beim Kompilieren wird der vom Entwickler geschriebene Quellcode mit Hilfe eines Compilers in Maschinensprache übersetzt. Falls noch nicht installiert, benötigen wir die Pakete *checkinstall* und *build-essential*. Die zum Kompilieren notwendigen Befehle lauten: *configure*, *make* und *make install*. Der Installationsprozess findet per Terminal statt.

Das detaillierte Erklären des Vorgangs würde den Rahmen dieses Artikels sprengen. Vielleicht berichten wir in einer der nächsten Ausgaben von Yalm ausführlich darüber. Bis dahin verweisen wir auf Quellen im Internet. [7] [8]

Jürgen Weidner
joschi@yalmagazine.org

Informationen

[1] Erklärungen zur grafischen Paketverwaltung
http://wiki.ubuntuusers.de/Synaptic

[2] Erläuterungen zur Paket- verwaltung unter KDE
http://wiki.ubuntuusers.de/Adept

[3] Definition der Repositories
www.yalmagazine.org/link/43

[4] Spezifizierung der Paketverwaltung
http://wiki.ubuntuusers.de/Paketverwaltung

[5] Begriffsklärung von APT
http://de.wikipedia.org/wiki/Advanced_Packaging_Tool

[6] Allgemeine Anleitung zum Installieren von Programmen
http://wiki.ubuntu-forum.de/index.php/Programme_installieren

[7] Sekundärsoftware mittels Paketquellen
http://www.yalmagazine.org/link/43

[8] Vorgehensweise beim Kompillieren von Programmen
http://wiki.ubuntuusers.de/Programme_kompilieren

Die Qual der Wahl

Wenn man sich für Linux als Betriebssystem für seinen Rechner entscheidet, hat man die Möglichkeit, aus einer Vielzahl von Desktop-Oberflächen zu wählen. In diesem Artikel möchten wir drei prominente grafische Oberflächen vorstellen.

Um mit Linux arbeiten zu können, benötigen wir genau genommen nur eine Konsole, mit deren Hilfe wir per Texteingabe mit dem Computer kommunizieren. Da sich heutzutage aber nur noch hartgesottene Freaks mit dieser Lösung zufriedengeben, darf es für die allermeisten Nutzer schon etwas Nettes fürs Auge sein.

Betrachten wir zunächst den typischen Aufbau einer Linux-Distribution: Als Fundament finden wir hier den Kernel [1], sozusagen den Motor des Ganzen. Darüber liegt das X-Window-System [2] und zuoberst, für uns sichtbar, der Windowmanager [3] bzw. der Desktop. Windowmanager sind schlanke, wenig Resourcen benötigende Desktop-Alternativen. Allerdings verfügen sie nicht unbedingt über einen Dateimanager [4] oder andere Verwaltungsprogramme, können aber damit nachgerüstet werden.

Dieser Aufbau gibt uns die Möglichkeit, gleichzeitig mehrere verschiedene Windowmanager und Desktops zu installieren und beim Einloggen in das System zu entscheiden, mit welchem wir arbeiten wollen.

Aber der Reihe nach. Zunächst nehmen wir die einzelnen Oberflächen, auch Desktopumgebungen genannt, unter die Lupe.

GNOME

Das *GNU Network Object Model Environment* [5] wurde 1997 gegründet und ist zur beliebtesten grafischen Oberfläche für Unix und Linux avanciert. Bei Ubuntu ist es der Standard-Desktop.

Die Entwickler von GNOME legten Wert auf einfache Bedienung, Stabilität des Systems und einwandfreies Funktionieren der Software. Außerdem sollte die Nutzung des Desktops intuitiv möglich sein und keine besonderen Kenntnisse voraussetzen.

GNOME wurde auf Basis des *Gimp-Toolkits GTK+* entwickelt und existiert derzeit in der Version 2.23. Die Hardware-Anforderungen sind eher mittelmäßig, mit einer CPU mit 1 GHz Taktfrequenz und ab 384 MB RAM ist man gut gerüstet.

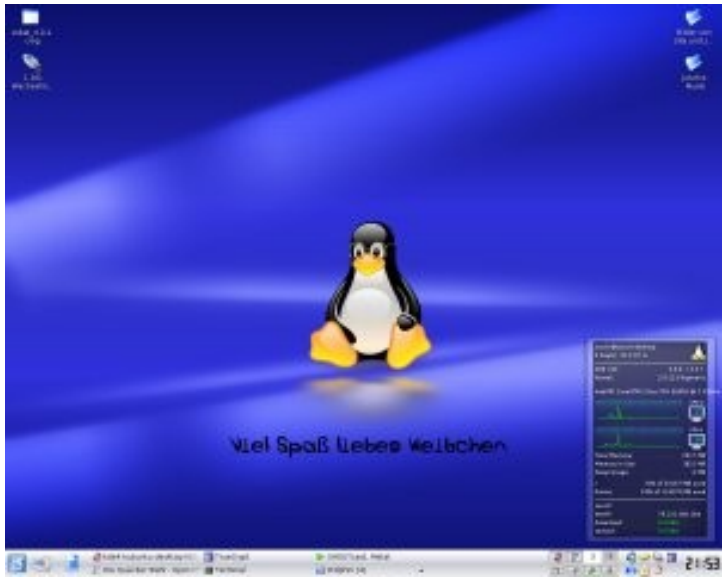
Die Qual der Wahl: Gnome...

Der Desktop ist leicht zu handhaben und läuft absolut stabil. Seit der Ubuntu Version 7.10 sind sehenswerte 3D-Anwendungen integriert. Seit GNOME 2.20 kommen sogar einige nette Effekte sofort mit GNOME Mit, sodass die Distributionen nichts eigenes mehr machen müssen, trotzdem bleibt Compiz sehr beliebt. Um diese nutzen zu können, werden jedoch höhere Anforderungen an die Ausrüstung gestellt. So muss die Grafikkarte 3D-fähig sein, 3D muss unter Linux funktionieren und die CPU muss mehr arbeiten.

Ein Versions-Update erfolgt alle 6 Monate, das System wird außerdem durch das Angebot von Updates immer auf dem Laufenden gehalten.

KDE

Das *Kool Desktop Environment* [6] (heute fast nur noch *K Desktop Environment*)wurde ein Jahre vor dem ersten Release von GNOME eingeführt.



...KDE...

Da KDE den proprietären *Qt Toolkit* von Trolltech als Basis hat, war die Akzeptanz der Linux-Gemeinde zunächst nicht sonderlich hoch. Seit April 1999 gab Trolltech aber auch seine QT-Bibliotheken frei, und seitdem genügt auch der KDE-Desktop den Ansprüchen der Nutzer an freie Software. KDE ist der Standard-Desktop von Kubuntu.

Während GNOME der beliebteste Linux-Desktop ist, schlägt Linus Torvalds [7], der Initiator von Linux, den Einsatz von KDE vor. Es gefällt ihm nicht, dass GNOME-Entwickler, um den Desktop einfach und übersichtlich zu gestalten, notfalls Funktionen weglassen oder gar wieder entfernen, um den Anwender nicht zu überfordern. Und in der Tat sind die Einstellungsmöglichkeiten beim KDE-Desktop größer. Hier werden dem Benutzer mehr Freiheiten gegeben, die einfache Handhabung leidet darunter jedoch nicht. Man sagt KDE nach, für Windows-Nutzer der am besten geeignete Linux-Desktop zu sein. Umsteiger finden sich sofort in ihm zurecht, eine größere Eingewöhnung ist nicht nötig.

KDE gibt es derzeit in zwei aktuellen Versionen. KDE 3.5x, eine verlässliche, aber nun nicht mehr ganz aktuelle Version, und KDE4, die noch nicht ganz stabile, aber zukunftsweisende Neuentwicklung.

...oder doch XFCE?

Die Systemvoraussetzungen liegen auch hier bei 1 GHz Taktfrequenz für die CPU, der Arbeitsspeicher sollte jedoch nicht kleiner als 512 MB sein, wenn es Spass machen soll.

Was 3D-Funktionen und Updates betrifft, so gilt hier dasselbe wie bei GNOME.

XFCE

Auch die Entwicklung des *XForms Common Environment* [9] begann im Jahre 1996. Diese GUI wendet sich insbesondere an die Nutzer älterer Computer oder an diejenigen, die es gern etwas schneller haben. Wie GNOME basiert XFCE auf dem *Gimp Toolkit*, die aktuelle Version trägt die Bezeichnung 4.4. Die Hardwareanforderungen sind wahrhaft spartanisch, ab einer 300 MHz CPU läuft es rund, 128 MB Arbeitsspeicher genügen.

Wo findet man, außer bei Linux, ein modernes Betriebssystem mit an-

Hardwareanforderungen

Distribution	CPU Taktfrequenz ca.	RAM MB
Ubuntu / Gnome	1,0 GHz	384
Kubuntu / KDE	1,0 GHz	512
Xubuntu / XFCE	0,3 GHz	128

Wie immer gilt: Mehr ist besser, aber auch mit etwas weniger läuft es noch, natürlich mit eingeschränkter Performance.

sprechendem Desktop und solch geringen Anforderungen? Nach zwölfjähriger Entwicklungszeit ist es eigentlich überflüssig darauf hinzuweisen, dass uns hier ein sicheres und stabiles System zur Verfügung steht.

Transparenz, Schatten und selbst 3D-Effekte sind mit XFCE machbar. Auch Upgrades und Updates werden in gewohnter Manier geliefert.

Wer sich nun noch nicht entscheiden kann, sollte sich beispielsweise auf Youtube anschauen, was man aus den vorgestellten Desktops machen kann.

Alternativ können wir uns natürlich die entsprechende Distribution herunterladen, als ISO-Image auf CD brennen und mit den Live-CDs testen, was uns am besten gefällt. Hier kann man auch andere Distributionen testen, die die eine Desktop-Umgebung vielleicht besser integrieren. So finden viele, dass KDE unter OpenSuse besser läuft als unter Kubuntu. Man kann auch, entsprechende Hardware vorausgesetzt, alle hier vorgestellten Desktops parallel installieren und nach Lust und Laune mit ihnen arbeiten. Hierbei wird sich wohl schon bald ein Favorit herausstellen.

Die Installation einer anderen grafischen Oberfläche ist unter Linux so einfach wie die Installation anderer Software: Wir suchen in unserer Paketverwaltung den gewünschten Desktop aus, installieren ihn und können zukünftig bei der Anmeldung am Login-Fenster entscheiden, mit welcher Variante wir arbeiten wollen. Alternativ ist auch die Installation per Terminal möglich, beispielsweise holen wir uns den XFCE-Desktop mit dem Kommando

```
sudo apt-get install →
xubuntu-desktop
```

Da sich die hier vorgestellten Oberflächen an den Vorgaben der freedesktop.org orientieren, funktioniert die Nutzung von Daten, unabhängig davon welche GUI genutzt wird, problemlos. Das bedeutet, dass Dateien, die in Ubuntu angelegt wurden auch in Xubuntu oder Kubuntu genutzt werden können – es gibt hier keine Unterschiede zwischen den Desktops.

Jürgen Weidner
joschi@yalmagazine.org

Informationen

[1] Erläuterungen zum Linux-Kernel
http://de.wikipedia.org/wiki/Linux_Kernel

[2] Erörterungen zum netzwerkfähigen Fenstersystem
X11http://www.chemie.fu-berlin.de/glossar/x11.html

[3] Artikel der Wikipedia zu Fenstermanagern
http://de.wikipedia.org/wiki/Fenstermanager

[4] Auflistung bekannter Dateimanager
http://www.tuxfutter.de/wiki/Linux:Dateimanager

[5] Internetpräsenz der Desktopumgebung GNOME
http://www.gnome.de/

[6] Die Geschichte von KDE
http://wiki.ubuntuusers.de/Geschichte_von_KDE

[7] Werdegang von Linux Torvalds
http://www.linux.de/linux/linus_torvalds.php3

[8] Artikel über XFCE
http://wiki.ubuntuusers.de/ Xfce

gDesklets

Wolltet ihr euch schon immer mal den Wetterbericht, Newsticker oder Systeminformationen anzeigen lassen, ohne gleich ein neues Fenster öffnen zu müssen? - gDesklets macht's möglich.

gDesklets ist ein in PyGTK, einem Python-Wrapper für GTK+, geschriebenes Programm, mit dem dem man sich sogenannte Desklets, auch als Widgets oder Gadgets (bei Vista) bekannt, direkt auf dem Desktop anzeigen und beliebig positionieren kann, ohne ein extra Fenster zu öffnen.

Obwohl gDesklets ursprünglich für GNOME entwickelt wurde, ist es, die entsprechenden Bibliotheken vorausgesetzt, auch auf anderen Desktop-Umgebungen lauffähig. Achtung: Viele Bibliotheken können aber das System ausbremsen...

Ein Desklet besteht im Wesentlichen aus einer Anzeige (Display) und (optional) aus einem Control. Das Display besteht aus einer Datei mit der Endung .display, in der das eigentliche Desklet gespeichert ist. Da dieses aber in einer Sandbox läuft, hat es keinen Zugriff auf das System. Dafür sind die Controls da; ein Control lässt sich ganz einfach in der Display-Datei einbinden und so auf dessen Funktionen zugreifen.

Zuerst installieren wir mit

```
sudo apt-get install →
gdesklets gdesklets-data
```

gDesklets und dazu eine Auswahl von Desklets. Nun finden wir gDesklets unter «*Anwendungen - Zubehör - gDesklets*».

Zuerst sollten wir mit

```
gdesklets configure
```

das Konfigurationsfenster öffnen und einige Einstellungen vornehmen.

Im 1. Punkt müssen wir den Editor wählen, mit dem wir uns Quelltexte von Desklets anzeigen lassen wollen (näheres dazu im 2. Teil dieses Artikels). Da *gnome-text-editor* meist auf *gedit* verweist, können wir diese Einstellung belassen, falls wir mit gedit arbeiten wollen. Andernfalls geben wir unseren bevorzugten Texteditor an.

Damit die Desklets in der richtigen Größe angezeigt werden, müssen wir die DPI (Dots per Inch = Punkte pro Zoll)-Zahl anpassen. Entweder messen wir direkt z.B. mit einem Lineal den Balken nach oder geben in einem Terminal

```
xdpyinfo | grep resolution
```

ein und verwenden den ersten Wert aus der Ausgabe.

Die nächsten beiden Konfigurationsmöglichkeiten sind selbsterklärend. Beim letzten Punkt gibt der Schwebemodus an, ob das Desklet immer im Vordergrund stehen soll oder nicht.

Nun öffnen wir mit

```
gdesklets shell
```

die gDesklets Shell, in der man Desklets installieren und starten kann.

Um ein Desklet zu starten, klickt man *Displays* an und wählt im Menü «*Datei - ausgewähltes Desklet starten*». Alternativ kann man z.B. Desklets aus dem Internet mit der Funktion «*Desklet (nicht-lokal) starten*» aufrufen. Neue Desklets installieren wir mit *«Paket installieren»* bzw. *«Nicht-lokales Paket installieren»*. Finden können wir solche Pakete z.B. auf der Seite des Projekts [1] oder im *«unofficial gDesklets Desklet Archive»* [2].

Was aber tun, wenn es das gewünschte Desklet nicht gibt? Ganz einfach: Selber programmieren. Wie das geht wird jetzt erklärt:

Programmierung von eigenen Desklets

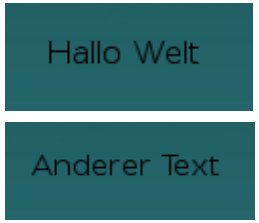
Das Display eines Desklets basiert auf einer XML(**X**-tensible **M**arkup **L**anguage)-Datei. In XML werden das Aussehen, die Konfigurationsdaten (Rechtsklick auf ein Desklet -> Desklet konfigurieren) und die Funktionalität des Desklets beschrieben.

Das oberste Tag ist *<display>*, in das alle anderen Tags eingeschlossen sind. Ihm untergeordnet sind *<meta>*, das z.B. den Autor und eine Beschreibung des Desklets beinhaltet, *<control>*, über das man ein Control einbetten kann, *<prefs>*, in dem man die Einstellungsmöglichkeiten beschreibt (die man dann später im Kontextmenü des Desklets unter dem Punkt *«Desklet konfigurieren»* aufrufen kann), *<script>*, in dem man das Desklet mit Python steuern kann und allerlei andere Tags, mit denen das Aussehen des Desklets bestimmt wird. Außer *<meta>* und *<prefs>* gehören alle zur sogenannten ADL *(Applet Description Language)*.

Zunächst wollen wir einen Text anzeigen lassen, der bei einem Mausklick ausgetauscht wird:

```
#Gibt an, dass ein UTF-8 kodiertes XML-Dokument folgt

<?xml version="1.0" → encoding="UTF-8"?>

#Leitet das Display ein, die Flags «sticky» und «below» geben an, dass das Desklet auf allen Arbeitsflächen und immer im Hintergrund ist

<display window-flags=→ "sticky, below">

   #Das Label; auf Mausklick wird die Funktion klick() aufgerufen

   <label on-click="klick()" → id="text" value="Hallo Welt" → font="Sans 12" />

   #Das Python-Skript; hier ist die Funktion klick() definiert, die den Text ändert

   <script><![CDATA[

   def klick():

      #Hier wird der Text je nach aktuellem Wert verändert; mit «Dsp» kann auf die Elemente zugegriffen werden

      if Dsp.text.value == → "Hallo Welt":

         Dsp.text.value = → "Anderer Text"

      elif Dsp.text.value == → "Anderer Text":

         Dsp.text.value = → "Hallo Welt"

      ]]>

   </script>

#Ende des Displays

</display>
```

Dieses XML-Dokument speichern wir jetzt als *«hallowelt.display»* im Verzeichnis *«~/.gdesklets/Displays»*. Nun öffnen wir die gDesklets-Shell, klicken *Display* an, dort dann auf *uncategorized* und starten das Desklet *hallowelt*. Nun sollten wir ein Desklet sehen, was auf Mausklick den Text ändert.



Austausch von Texten

Um jetzt die Fähigkeiten eines Desklets ein bisschen zu erweitern, verwenden wir ein Control. Ein typisches Control besteht aus einer Interface-Datei, die laut Konvention mit einem »I« beginnt, und einer Datei mit dem Namen *«__init__.py»*. Die Programmierung erfolgt wie bei den *Displays* mit Python.

Die Interface-Datei ist die Schnittstelle der Displays zu den Controls. Hier werden die von der Schnittstelle bereitgestellten Properties (Eigenschaften) und ihre Rechte festgelegt. Dazu werden die entsprechenden Klassen aus der «libdesklets» inkludiert. Properties [3] können Methoden zugewiesen werden, die automatisch beim Lesen, Schreiben oder Löschen der Property aufgerufen werden. Außerdem kann man eine Beschreibung ihres Inhalts definieren.

In diesem Beispiel darf *«Beispielproperty»* gelesen werden, andere mögliche Werte sind *«WRITE»* und *«READWRITE»*.

```
from libdesklets.controls → import Interface, Permission

class IBeispiel(Interface):

   Beispielproperty = → Permission.READ
```

In der *«__init__.py»*-Datei wird die eigentliche Funktion des Controls festgelegt:

Zuerst werden die Basis-Control-Klasse, die elementare Funktionen für jedes Control enthält, und die Interface-Datei (oder auch mehrere Dateien) inkludiert. Dann wird eine Klasse mit dem Namen des Controls definiert, die sich von der Interface- und Basis-Control-Klasse ableitet und eine *«__init__»*-Methode beinhaltet, in der das Control wiederum durch die *«__init__»*-Methode der Basis-Control-Klasse initialisiert wird. Danach können weitere Methode folgen, die die Funktionalität des Controls bilden. Am Ende wird noch eine Funktion *«get_class()»* definiert, die den Namen eben dieser Control-Klasse zurückgibt.

```
from libdesklets.controls → import Control

from IBeispielInterface → import IBeispielInterface

class BeispielControl(Cont→ rol, IBeispielInterface):

def __init__(self):

Control.__init__(self)

def get_class(): return → BeispielControl
```

Nun wollen wir zum Schluss ein kleines Beispiel-Desklet programmieren, das sich einem Control bedient. Es soll jeweils Datum und Uhrzeit präsentieren.

Zuerst die *«__init__.py»*-Datei:

```
#Das time-Modul wird inkludiert

import time

from libdesklets.controls → import Control

from IUhrzeit import IUhrzeit

class Uhrzeit(Control, → IUhrzeit):

   def __init__(self):
```

```
      Control.__init__(self)

      #Ein Timer wird
initialisiert, der jede
Sekunde die Methode __erneuern
aufruft

      self._add_timer(1000, →
self.__erneuern)

   #Die Methode __erneuern
«updatet» die Property
«datum_und_zeit»

   def __erneuern(self):


self._update("datum_und_zeit")

          return True

   #Datum und Zeit werden
ermittelt

   def __get_zeit(self):

      self.__lokalzeit = →
time.localtime()

      return self.__lokalzeit

   #Die Property
«datum_und_zeit» wird
angelegt; sie bekommt die
zuständige «Getter»-Methode
und eine Erklärung ihres
Wertes zugewiesen

   datum_und_zeit = →
property(__get_zeit, doc = →
"Datum und Uhrzeit")

def get_class(): return →
Uhrzeit
```

Nun die Interface-Datei, die nicht allzu groß ausfällt:

```
from libdesklets.controls →
import Interface, Permission

class IUhrzeit(Interface):

   datum_und_zeit = →
Permission.READ
```

Und zum Schluss das Display:

```
<?xml version="1.0" →
encoding="UTF-8"?>

<display window-flags=→
"sticky, below">

#Das Control wird unter dem
Namen «uhrzeit» eingebunden;
über diesen Namen kann jetzt
auf es zugegriffen werden

<control id="uhrzeit"
interface="IUhrzeit:→
cway7qkzxzjnybi0qtwukmy1k-2"/>

   <label id="Zeit" →
font="Sans 12" />

   <script><![CDATA[

   #Die Funktion aktualisieren
formatiert die übergebenen
Werte und stellt sie dar

   def
aktualisieren(new_value):

      Dsp.Zeit.value = →
"%02d. %02d.%d %02d:%02d:→
%02d" % (new_value[2], →
new_value[1], new_value[0], →
new_value[3], new_value[4], →
new_value[5])

   #Die Property
«datum_und_zeit» wird an die
Funktion «aktualisieren»
gebunden; jedesmal, wenn sich
der Wert der Property ändert,
also jede Sekunde, wird die
Funktion aufgerufen

    uhrzeit.bind("datum_und_→
zeit", aktualisieren)

       ]]>

   </script>

</display>
```

Um dieses Desklet jetzt zum Laufen zu bringen, kopieren wir die beiden Control-Dateien nach *~/.gdesklets/Controls/Uhrzeit* und das Display z.B. unter dem Namen *uhrzeit.display* nach *~/.gdesklets/Displays*.

Das Ergebnis unserer Mühen: eine Uhr für den Desktop

Nun müssen wir eventuell noch die *Interface-ID* in der Display-Datei anpassen. Dazu öffnen wir die gDesklets Shell, wechseln auf den Reiter *Controls* und kopieren die ID in Form von *«Name des Interface:Checksumme»* in unsere Display-Datei. Nun können wir das Desklet starten.

Das war jetzt nur ein kleiner Einblick in die Möglichkeiten von gDesklets. Bei der Programmierung von Desklets sind einem dank Controls (fast) keine Grenzen gesetzt. Wer tiefer in die Materie einsteigen will, dem sei das gDesklets-Entwicklerbuch [4] empfohlen.

Alle Listings dieses Artikels sind zusätzlich auch im Yalm-Forum [5] abrufbar.

Lauris Ding
luxi@yalmagazine.org

Informationen

[1] Die Projektseite
http://www.gdesklets.de/

[2] The unofficial gDesklets Desklet archive
http://gdesklets.zencomputer.ca/

[3] Dokumentation der property()-Funktion
http://docs.python.org/lib/built-in-funcs.html#l2h-57

[4] Das gDesklets-Entwicklerbuch
http://develbook.gdesklets.de/

[5] Listings dieses Artikels
http://www.yalmagazine.org/forum/showthread.php?tid=527

Tipps und Tricks für die Shell (2)

Im zweiten Teil der Serie geht es wieder um Nützliches und Wissenswertes rund um die Kommandozeile. Es geht unter anderem um das Suchen mit Grep, Ausführen von Scripten, das Rechnen mit der Bash und am Ende wartet noch ein kleines Rätsel auf seine Lösung.

Zuerst noch ein Wort zur letzten Ausgabe, dort haben wir über den Einsatz von *«sudo –s»* zum Erlangen von root-Rechten berichtet. Leider hat sich in die Beschreibung ein Fehler eingeschlichen, der unserem Leser Nils R. nicht entgangen ist: Bei der Verwendung von «sudo –s» wird die Variable $HOME nicht an root angepasst und behält den vorherigen Wert. Vielen Dank für den Hinweis.

Grep - mehrere Zeilen ausgeben

Das Programm *grep*, für das übrigens auch Portierungen auf Windows existieren, eignet sich hervorragend zum zeilenorientierten Suchen nach Textmustern. Es gehört schon lange zu den absoluten Standard-Programmen für die Befehlszeile. Allerdings ist oftmals nicht nur die gesuchte Zeile interessant, sondern auch die vorangegangene oder nachfolgende. Hier steht man bei normaler Verwendung von Grep vor einem Problem: Grep wendet das Suchmuster ausschließlich auf die aktuelle Zeile an, und auch sogenannte *«Multiline-Expressions»* (also Suchbegriffe die von ihrer Beschaffenheit auf mehrere Zeilen gleichzeitig zutreffen können) führen nicht zum Ziel. Es ist trotzdem mit den Optionen –A (After) und –B (Before) möglich, eine bestimmte Anzahl an Zeilen vor und/oder nach der Fundstelle auszugeben:

```
$ grep "127.0.0.1" /etc/hosts
127.0.0.1 localhost
$ grep -A2 "127.0.0.1" → /etc/hosts
127.0.0.1       localhost
192.168.0.1     ubuntu
192.168.0.2     windows-desktop
```

Rechnen

Die Bourne-Shell war nicht in der Lage zu rechnen. Hier mussten externe Programme wie *«bc»* und *«expr»* eingesetzt werden. Die Bash hingegen bringt eine eingebaute Integer-Arithmetik mit und erlaubt so das unkomplizierte Rechnen mit ganzen Zahlen. Eine normale Addition in der Bash:

```
$ let zahl=1+1
$ echo $zahl
2
```

Die Verwendung eines eigenen Schlüsselwortes ist oft umständlich. Hier bietet die Bash noch eine weitere Möglichkeit. Mithilfe von doppelten Klammern wird automatisch in den Rechenmodus geschaltet:

```
$ zahl=$((1+1)); ((zwei=1+1))
$ echo $zahl
2
$ echo $zwei
2
```

C syntax for-schleife

In vielen Programmiersprachen sehen for-Schleifen / Zählschleifen fast identisch aus. Wer oft in einer Sprache der C-Familie programmiert, wird diese vielleicht beim Erstellen eines Bashscripts vermissen. Es ist mit einem einfachen Trick allerdings jedoch möglich, eine for-Schleife in fast gewohnter Syntax zu schreiben:

```
for ((i=0;i<10;i++)); do → echo $i; done
```

Diese Schreibweise bringt keinerlei Vorteile gegenüber der Standardsyntax:

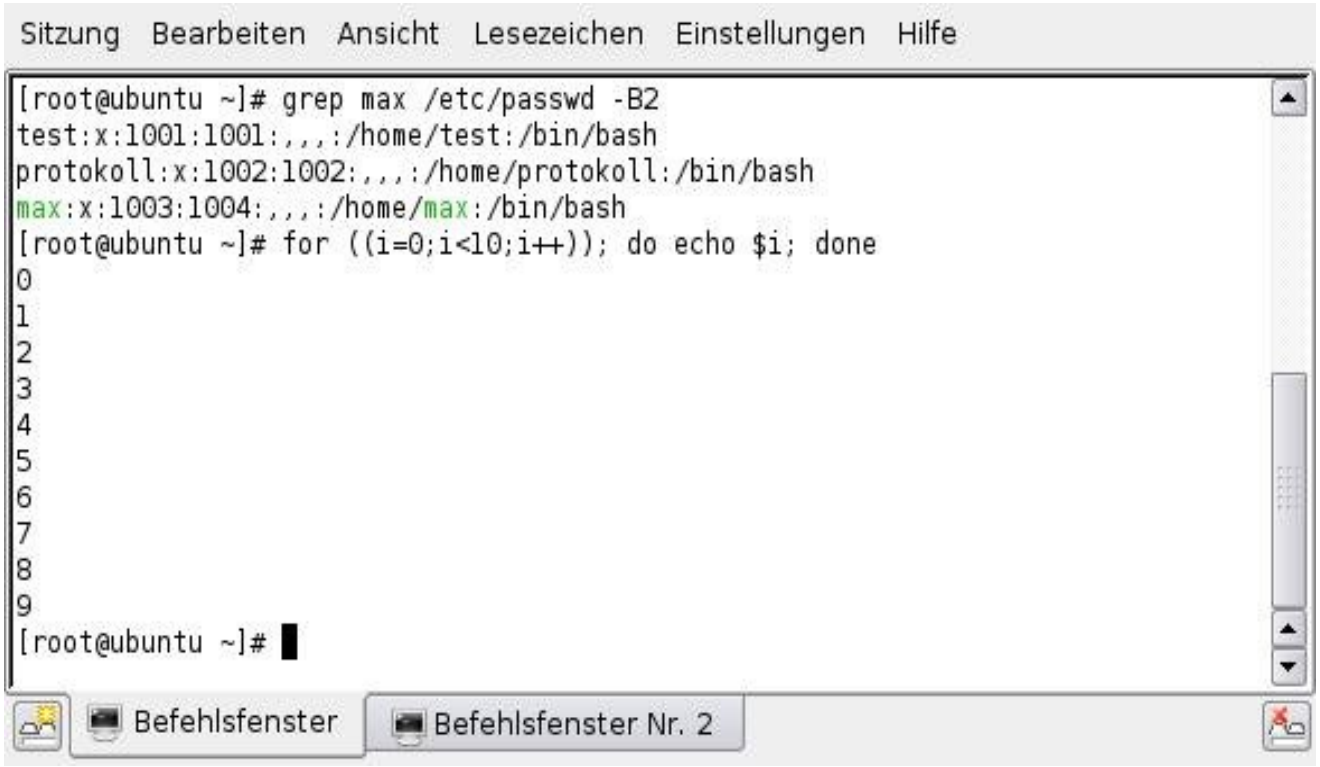
```
for i in {0..9}; do echo $i; → done
```

Punkt

Wer schon Shellscripte ausgeführt hat, wird es vielleicht wissen oder bemerkt haben: Jedes Script wird in einer sogenannten *«Subshell»* aufgerufen. Dies hat zur Folge, dass Änderungen, die im Script durchgeführt werden, keinen Einfluss auf die aufrufende Shell haben. Variablen, die gesetzt werden, sind später nicht in der aktuellen Shell verfügbar. Eigentlich ein gewünschtes und normales Verhalten. Die sogenannte *«Kommandosubstitution»* ist im Prinzip auch nur ein Aufruf einer Subshell:

```
$ var= $(date)
$ echo $var
Di 1. Jul 19:34:08 CEST 2008
```

Allerdings kann es von Zeit zu Zeit vorkommen, dass Scripte Änderungen in der aktuellen Umgebung vornehmen sollen. Um das zu erreichen, muss sichergestellt sein, dass keine Subshell gestartet wird und das Script im Kontext der Shell des Benutzers läuft. Beides kann mit einem



C-Syntax-ähnliche For-Schleife und Grep mit mehrzeiliger Ausgabe.

vorangestellten Punkt erreicht werden:

```
$ . myScript.sh
```

So wird das Script von der gerade laufenden Befehlszeile interpretiert und die Befehle darin ausgeführt.

Das funktioniert auch, wenn keine Ausführungsrechte vorhanden sind. Jeder, der auch nur Leserechte auf die Datei hat, kann das Script ausführen. Es wird immer im Kontext des Aufrufers abgearbeitet. Wenn dieser keine Rechte besitzt, eine bestimmte Aktion innerhalb des Scripts auszuführen, kann er diese Handlung auch weiterhin nicht durchführen. Das mag nun auf den ersten Blick wie eine große Sicherheitslücke erscheinen aber wenn man sich vor Augen führt, dass ein Script lediglich eine Folge von Kommandos ist, die der Benutzer auch von Hand eingeben könnte, wird schnell klar, dass es sich hier um ein vollkommen normales Verhalten handelt.

Daraus folgt: Ein Shell Script kann immer ausgeführt werden, wenn Leserechte vorhanden sind.

Rätsel

Am Schluss eines jeden Artikels dieser Serie wollen wir von jetzt an ein kleines Rätsel stellen. Die Thematik und Aufgabenstellung lehnt sich an die Tipps des aktuellen Artikels an.

Falls jemand die Antwort weiß, würden wir uns sehr über einige Zuschriften an max@yalmagazine.org oder redaktion@yalmagazine.org freuen.

Die Einsender der ersten drei richtigen und schlüssigen Antworten werden in jeder Ausgabe an dieser Stelle namentlich genannt und können auf Wunsch auch einen eigenen kurzen Gruß an die Welt anbringen lassen. Wer lieber anonym bleiben möchte, ist natürlich trotzdem herzlich eingeladen eine Antwort zu schreiben. Diese werde werden wir dann selbstverständlich ohne Namen in die Liste eintragen. Wir hoffen auf viele Antworten.

Hier nun das Rätsel der aktuellen Ausgabe:

```
$ let question=$((zahl=5); → 
echo "1$zahl")-5

$ echo $question

-4
```

Warum lautet das Ergebnis *«-4»* ? Ein kleiner Tipp: Die Klammern haben mit der Rechnung nichts zu tun.

Maximilian Schnur
max@yalmagazine.org

Bunte Seite

Chaosradio Express Podcast zu Python und Pypy

Im Chaosradio Express [1] gab es mal eine Sendung über Python und Pypy welche sehr interessant und empfehlenswert ist. Es wird darüber diskutiert, was an Python gut und was schlecht ist. Auch wird das Pypy Projekt [2] vorgestellt.

Gimp Tutorials

Auf Gimpusers.de [4] gibt es viele spannende Tutorials zu Gimp. Außerdem gibt es dort auch ein paar Videotutorials.

Gameboy Spiele auf Linux

Visual Boy Advance [3] ist ein Gameboy Advance Emulator, mit welchem es möglich ist, Gameboy ROM's (Spielemodule) auf dem Computer zu spielen. Um Visual Boy Advance zu installieren, öffnet man ein Terminal und gibt

```
  sudo apt-get install →
visualboyadvance
```

ein. Die benötigten Spielemodule sind nur dann legal, wenn man das Spiel erworben hat.

Um nun ein Spiel im Emulator zu spielen benötigt man ein ROM. Das ROM öffnet man über das Terminal mit dem Befehl:

```
vba --throttle=100 --filter-→
super-eagle ROMNAME.gba
```

Wenn man die Spielemodule nicht über das Terminal starten will, kann man sich auch eine gtk-Version von Visualboyadvance installieren.

Nautilus Aktionen

Wenn man viele Bilder verkleinern will hat man zwei Optionen: Entweder man macht das ganze im Terminal oder man öffnet jedes einzeln mit einem Bildbearbeitungsprogramm und verkleinert es. Wäre es nicht schön, wenn man die Bilder markiert und im Kontextmenü auswählen kann, wie das Bild verkleinert werden soll? Es gibt eine Lösung: Nautilus Aktionen !

Als erstes muss man das Paket ***nautilus-actions*** aus dem Universe Repository installieren. Es werden bereits ein paar Aktionen mit installiert, z.B. um Dateien als Root oder den aktuellen Ordner im Terminal zu öffnen.

Weitere Aktionen könnt ihr unter [5] herunterladen. Die jeweilige **.schema*-Datei wird im Nautilus-Aktionen-Konfigurationsdialog unter «*System – Einstellungen – Nautilus-Aktionen*» importiert. Falls zusätzliche Dateien herunterzuladen sind, gibt es dazu eine eigene Erklärung. Es können auch selbst Aktionen geschrieben werden; Anleitungen dazu findet man unter [6].

Daniel Uhl
tuxfreak@yalmagazine.org

Informationen

[1] Website des Chaosradio-Express
http://chaosradio.ccc.de/cre088.html

[2] Homepage des Pypy Projekts
http://codespeak.net/pypy/dist/pypy/doc/home.html

[3] Internetauftritt des VisualBoyAdvance
http://vba.ngemu.com/

[4] Webangebot der Gimpusers
http://gimpusers.de

[5] Downloadangebot weiterer Nautilus-Aktionen
http://www.grumz.net/index.php?q=configlist

[6] Leitfaden zur Vorgehensweise
http://www.grumz.net/?q=taxonomy/term/10/9

Die Qualen der Remuids (II)

Dies ist der zweite Teil der Erzählung. Den ersten Teil kann man in der letzten Ausgabe nachlesen.

Silvia seufzt.

Krieg! Sie hasst Krieg, aber sie hat keine Wahl. «Wenn ich hier rumtrauere» dachte sie, «wird das auch nicht besser!» Also begibt sie sich auf den Weg zu dem Projekt *querz*, wo sie sich mit anderen Remuids und dem Anrufer treffen will.

Dort angekommen ist sie zunächst alleine. Erst nach einiger Zeit kommen mehrere Remuids, keiner weiß mehr als sie. Also: Warten. Aber was hilft warten? Dann wartet man nur auf den eigenen Tod...

* * *

Tom Blood guckt minutenlang aus dem Fenster, dann geht er zu seinem Schreibtisch um sich noch einmal dem Brief zu widmen. Wieder lässt er ihn sich laut vorlesen:

23.04.2243

Sehr geehrter Minister Tom Blood,

Wir sind sehr enttäuscht und wütend. Ihre Remuids sind schön und gut. Sie funktionieren sogar gut - zu gut. Sie entwickelten ein Eigenleben! Sie werden fast wie Menschen. Und Menschen sind oft sehr eigensinnig. So wollen sie «Geld», Macht oder «Gegenstände» haben. Um das zu bekommen, leisten sie ihren Auftraggebern gewisse Dienste. Natürlich sind diese Dienste meist illegal, was den meisten Remuids aber egal sein kann. Wer kann ihnen schon was anhaben?

Uns sind mehrere Fälle bekannt, in denen sie Firmen ausspionierten. Firmengeheimnisse wurden einfach an Dritte weitergegeben. Somit verlieren die Remuids ihren Sinn: Zu helfen. Mittlerweile richten sie erheblichen Schaden an! Nachdem bekannt wurde, dass Goolge Yahoo als «verdammte Besserwisser» bezeichnete, waren alle Verhandlungen und die Kooperation zwischen den Firmen Geschichte.

Natürlich sind solche Bezeichnungen nicht in Ordnung, aber die meisten machen so etwas... Offiziell kann man ohnehin nichts nachweisen, weil Yahoo den Quellcode nie hergeben würde. So schadet es einfach den Firmen.

Wir planen daher sogenannte «Muids» zu entwickeln, die die Remuids zerstören. Da Remuids aber einen großen Vorteil bringen, kann man sie nicht einfach ausschalten. Muids müssen einfach oft eine Art Update der Remuids machen. So, dass sie eben kein Eigenleben mehr führen können. Technisch dürfte es also kein großes Problem darstellen.

Wie hoffen auf ihre Unterstützung, schließlich haben wir Sie auch oft genug unterstützt. Oder wären sie ohne uns noch Minister?

Wir warten auf Ihre Antwort.

Mit freundlichen Grüßen,

Vereinigung Wirtschaft fördernder Firmen

Tom weiß, dass er keine Wahl hat. Wenn er nicht zusagen würde, hätte die *Vereinigung Wirtschaft fördernder Firmen* einiges gegen ihn in der Hand. Wenn er zusagt und es herauskommt, wird er sich wieder einmal der Öffentlichkeit und seiner Partei stellen müssen. Vielleicht wäre das das Ende seiner Karriere.

Er weiß aber auch, dass die Remuids nicht sehr dumm sein können, nicht nach dem was sie bisher gezeigt haben. Also wird er wohl gegen seine selbst erschaffene *Roboter kämpfen* müssen. Er hat Milliarden dafür investieren müssen.

* * *

Plötzlich meldet der Anrufer sich in *querz* und stellt sich vor: Er heißt Jack und setzt sich für Menschenrechte ein. Er fordert Offenheit: «Wir leben im 23. Jahrhundert, schon vor 200 Jahren waren offene Programme besser, wie kann man nur Closed-Source entwickeln?»

Silvia meldet sich zu Wort: Sie will wissen wie sie nun vorgehen, wie sie die Muids bekämpfen können, wie wahrscheinlich ein Sieg ist und warum ihr Schöpfer, Tom Blood, gegen seine *Kinder* kämpft.

Leider kann Jack nicht alle Fragen beantworten. Warum Tom das tut kann man nur erahnen. Jack wollte keine Gerüchte aufkommen lassen. Aber er hat eine Strategie für den Kampf gegen die Muids: Überzahlangriff war das Stichwort. Verluste muss man dabei wohl hinnehmen.

«Formiert euch in Vierer-Gruppen», schreibt er, «und verteilt euch. Ab und zu kann ich euch eine Karte liefern, hier eine, die schon viele von euch kennen. Blaue Punkte sind Remuids und rote Muids. Das grüne

sind Datenverbindungen, die ich hier sicher keinem weiter erklären muss.»

Jack meint abschließend, dass Muids nicht zu erkennen seien. Aber Remuids würden bemerken, wenn sie angegriffen werden. Dann sollten sie versuchen, die Quelle des Angriffs auszumachen und zu vernichten. Schließlich sollten sie in *querz* mitteilen wie sie es geschafft hatten einen Muid zu zerstören. Wenn sie nicht angegriffen werden, sollten sie sich unauffällig verhalten.

Silvia schließt sich mit Pete, Angelo und Rosa zusammen.

Lange passiert nichts. Die Vier fangen an sich zu fragen, worauf die Muids noch warten. Warum haben sie nicht längst zugeschlagen? Warten sie, bis sie noch mehr sind? Oder ist es nur Zufall, dass sie nicht entdeckt worden sind? In *querz* vorbeischauen ist auch nicht möglich; wenn sie zu oft *dort* sind, würde es auffallen. Also bleibt nur warten...

Auf einmal spürt Silvia einen Stich. Sie versucht zu schreien, ist aber zu geschockt um auch nur einen Laut herauszubringen. Sie denkt schon, dass es vorbei ist. Aus. Ende.

Noch ein Stich. Und noch einer. Wenn sie ein Mensch wäre, würde sie nun sicher ohnmächtig werden, doch sie muss die Schmerzen aushalten.

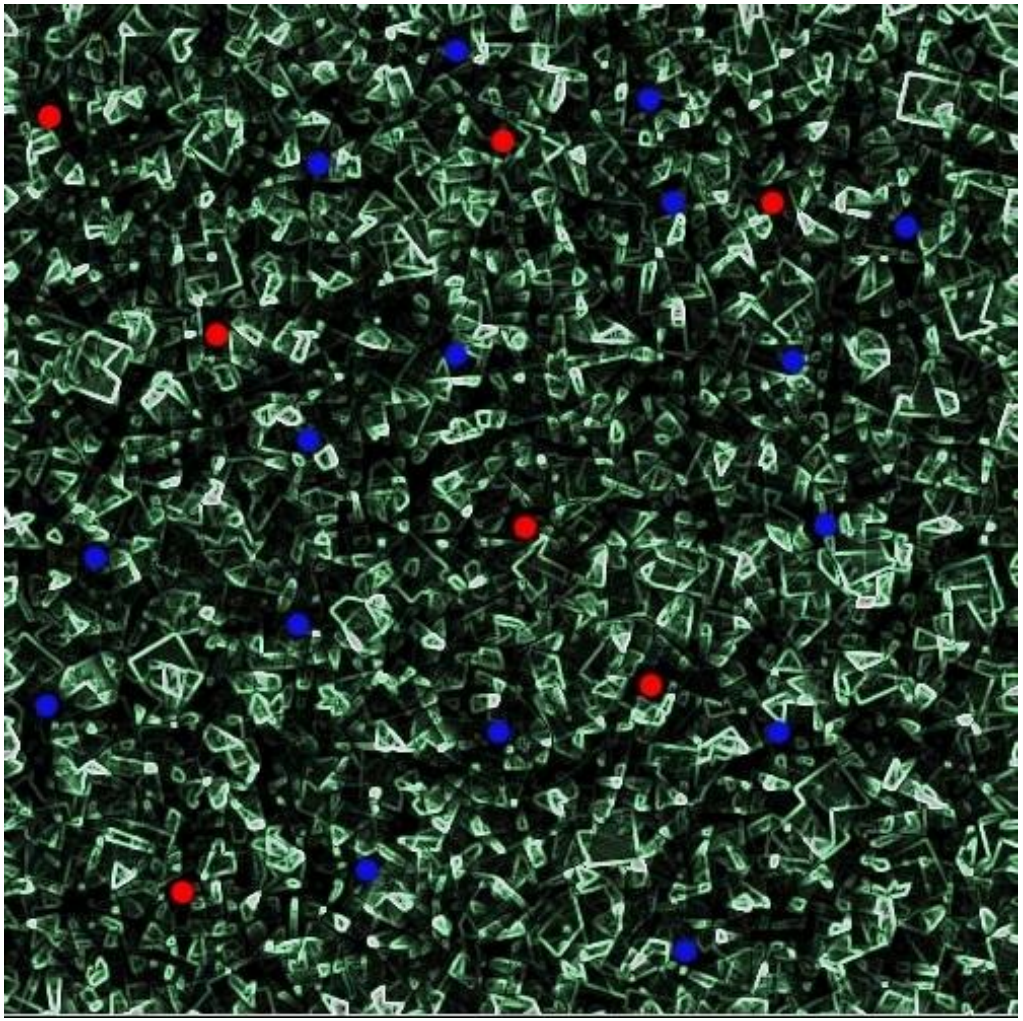
Sie merkt, dass sie noch Töne von sich geben kann! Ein Versuch ist es wert: Kurz, Kurz, Kurz, Lang, Lang, Lang, Kurz, Kurz, Kurz. SOS. Mehrmals sendet sie das bekannte Morsezeichen in der Hoffnung, Pete, Angelo oder Rosa würden sie verstehen.

Wieder ein Stich. Nun fehlt ihr sogar die Kraft Signale zu geben.

Doch sie hat Glück: Angelo bemerkt sie: «Oh, mein Gott!» *schreit* er. Nun werden auch Pete und Rosa aufmerksam. «Silvia, Silvia!»

Rosa wühlt in den Einsen und Nullen, sucht nach einem Anhaltspunkt: Irgendwo muss doch so ein Muid sein. Pete fängt an ihr zu helfen, hier geht es um Leben und Tod, doch sie müssen sich konzentrieren.

Angelo schreit auf. Er wünscht sich, lieber tot zu sein, doch er hat keine Wahl. Rosa schafft es nicht sich zu konzentrieren. Noch nie zuvor musste sie so arbeiten. Nervös versucht sie einen Muid zu finden. Doch es will ihr nicht gelingen.



Eine (nicht aktuelle) Karte, die zeigt wo Muids (rot) und Remuids (blau) sind

Pete schreit auf, doch er wurde nicht angegriffen: «Da, Muids! Ich kann ihren Code sehen!»

«Veränder oder lösch ihn! Schnell!»

«Ich kann nicht, es geht nicht.»

«Das muss gehen! Verdammt, das muss doch gehen.»

«Nein, ich hab keine Rechte.»

«Wie? Keine Rechte? Du *musst* Rechte haben»

«Dachte ich auch...»

Auch Pete vernimmt ein Stechen. Er weiß, dass es vorbei ist. Eine Chance hatte er nie gehabt. Eine Gehirnwäsche stand an. Mit letzter Kraft flüsterte er seine letzten Worte: «Rosa, Rosa, du musst Jack sagen was... was... pa-passiert ist.» Dann hörte man auch von ihm keinen Laut mehr.

Rosa weiß nun nichts mehr. Nicht einmal ob Silvia, Angelo und Pete noch leben oder bereits tot sind. Sie weiß, dass sie schnell weg *gehen* müsste, allerdings schafft sie es nicht. Wie gelähmt wartet sie darauf ein Stechen zu spüren.

Silvia *sieht* lange Zeit nichts mehr. Dann bemerkt sie eine Stimme. Nein, zwei oder mehr Stimmen. Sie versucht etwas zu erkennen: Sie findet sich unter hunderten von Remuids wieder. Alle sind ähnlich verwundert wie sie. Sie sind in einem großen, kahlen Raum. Stimmen werden irgendwie so umgeformt, dass sie sie verstehen kann.

Tom Blood tritt auf eine Bühne und verkündet seinen Sieg: «Hallo, da seid ihr alle!» Er lacht diabolisch und düster. Silvia bemerkt seine nur schlecht überspielte Nervosität.

«Tut mir ja leid, aber was sein muss, muss sein. Warum nehmt ihr illegale Aufträge an? Dann bleibt mir nichts als die Zerstörung. Ach, was sag ich: Keine Angst, morgen schon werdet ihr euch wieder frei bewegen können. Ihr wisst dann nur nicht mehr, wer ihr einmal wart.»

Silvia weiß, dass sie verloren hat, also kann sie auch dreist werden. Was hat sie denn schon noch zu verlieren?

«Hey, wie haben diese Muids uns fertig gemacht?»

Er lachte. «Ich dachte nicht, dass ihr so naiv seid. Weißt du wie lange man dafür braucht, so etwas zu entwickeln? Das ist viel zu aufwändig und vor allem zu teuer. Warum glaubst Du, daß ich das wissen könnte?»

«Wer oder was war es dann?»

«Kennst du Jack? Hast du dich nie gefragt, wo er die ganzen Informationen herhat? - Nun, ich sage es mal so: Er kämpft nicht für so etwas wie Menschenrechte.»

Alle im Raum sind schockiert, keiner ist fähig auch nur ein Wort zu sagen. Hatte Jack nicht versucht ihr Leben zu retten?

«Jack ist ein Bekannter von mir. Um seine Drogen finanzieren zu können brauchte er etwas Geld. Dafür macht er sich gerne seine Hände schmutzig."

Silvia weiß nicht, was schlimmer ist: Dass sie belogen wurde oder dass Minister in der EU solche Methoden anwenden.

«Einigen Remuids versprach er Macht, wenn sie ihm helfen. Kennst du Rosa noch, Silvia?»

Sie kann nicht antworten, gerade als sie sich vom Schock erholt hatte, stockte ihr wieder der Atem. Aber sie braucht gar nicht antworten, Tom weiß, dass sie sie kennt oder kannte.

«Sie und einige andere konnten euch Schmerzen zufügen, bis ihr zu nichts mehr in der Lage wart. Dabei konnten sie z. B. eine Folge aus Einsen und Nullen basteln, einige Remuids dachten so es seien Muids.»

Silvia merkt, dass es ihm Spaß macht *es* zu erzählen. In ein paar Stunden schon wird sie nichts mehr davon wissen.

Sie hat verloren. Endgültig.

Hier ein paar Einsen oder Nullen mehr oder weniger und es wird alles verändert. Sie besteht selbst nur aus den beiden Ziffern. Diese Gedanken basieren auf Einsen und Nullen. Was Einsen und Nullen in hundert Jahren anrichten können: Arme, arme Welt.

Mario Fuest
keba@yalmagazine.org

Schlusswort

Und nun findet auch diese Ausgabe ein Ende.

An dieser abschließenden Stelle möchten wir noch einmal um eure Mithilfe bitten: Zwar sind fast alle organisatorischen Schwierigkeiten beseitigt und fortführende Pläne in Arbeit, nicht zuletzt ein Grund dafür, dass ihr das Magazin pünktlich und in beachtlichen Umfang in Händen halten durftet, aber nichtsdestotrotz fehlt es uns noch immer an helfenden Händen. Unerheblich davon, ob ihr einen Artikel für das Magazin schreiben, euch um das Layout kümmern oder ein wenig der organisatorischen Last stemmen wollt. Jedes Hilfsangebot ist willkommen und wird gern gesehen.

Meldet euch bei Interesse bitte im Forum auf unserer Website, wo ihr weitere Informationen und erste Einweisungen erhaltet.

Doch auch wenn ihr uns nur eine Rückmeldung geben oder Verbesse rungsvorschläge unterbreiten wollt, ist das Forum neben einer Mail an redaktion@yalmagazine.org die beste Anlaufstelle.

Im Rahmen der letzten Ausgabe erhielten wir zwar einige Repliken, für welche wir uns an dieser Stelle herzlich bedanken möchten, aber bedauerlicherweise nicht genug, um eine Sektion der Leserbriefe für diese Ausgaben einrichten zu können. Dies beachtend, möchten wir euch um einige Reaktionen zu dieser Ausgabe bitten, sei es Lob, Kritik oder Anregungen, wir sind für alles dankbar.

In diesem Sinne wünschen wir euch alles Gute und hoffen, euch auch im nächsten Monat wieder als Leser von Yalm begrüßen zu dürfen.

Stefan Zaun
sciron@yalmagazine.org.

Das Yalm Team

Projektleitung
Tobias Kündig
tobias@yalmagazine.org

Mitwirkende dieser Ausgabe
Frank Brungräber
calexu@yalmagazine.org

Stefan Zaun
sciron@yalmagazine.org

Mario Fuest
keba@yalmagazine.org

Ralf Hersel
rhersel@yalmagazine.org

Maximilian Schnur
max@yalmagazine.org

Jürgen Weidner
joschi@yalmagazine.org

Angelo Gründler
speed@yalmagazine.org

Daniel Uhl
tuxfreak@yalmagazine.org

Lauris Ding
luxi@yalmagazine.org

Redaktion
http://yalmagazine.org/redaktion

Comics

Die Comics beziehen wir von *xkcd* [1]. Sie stehen unter der CC-BY-NC Lizenz [2]. Diese gestattet die Nutzung und Verbreitung für nichtkomerzielle Zwecke unter Nennung des Urhebers. In dieser Ausgabe haben wir auf Seite 12 einen Comic von Randall Munroe vorgestellt. Im Internet ist er unter [3] abrufbar.

Informationen

[9] http://xkcd.com/
[10] http://creativecommons.org/licenses/by-nc/2.5/deed.de
[11] http://xkcd.com/435/

Yalm 08/08 erscheint voraussichtlich am 15. August 2008