UpTimes

Mitgliederzeitschrift der
German Unix User Group

reStructuredText und Sphinx

Parallele Shell-Jobs

Zahlenspiele in der Shell

guug

2016-2

Inhaltsverzeichnis

Mütze auf, Schal um
Wintergruß aus der Redaktion

Manche sagen: Das Jahr 2016 war Mist. Das Gute ist ja, dass auch Schlechtes ein Ende hat. Das kommende Jahr 2017 hat es jedenfalls in sich: Die GUUG wählt einen neuen Vorstand, und die *UpTimes* sucht eine neue Chefredaktion.

von **Anika Kehrer**

Es gibt keine Sicherheit,
nur verschiedene Grade der Unsicherheit.

Anton Pawlowitsch Tschechow

In dieser Ausgabe finden wir uns ganz praxisnah wieder. Mehrere Aktivitäten füllen die Vereinsrubrik. Und sehr praktische Beweggründe stehen hinter den Sysadmin- und Software-Artikeln: Lenz Grimmer berichtet, mit dem starken Duo aus der Markup-Sprache *reStructuredText* und ihres Build-Frameworks *Sphinx* der Dokumentation in Software-Projekten Herr zu werden.

Mathias Weidner skriptet einen Ring, sie alle zu binden – die Jobs in der Shell nämlich, die gefälligst brav neben- und nacheinander ohne Zeitverschwendung starten. Er macht das einfach mittels Pipes. Dazu haben ihn Jürgen Plates „Shellskripte mit Aha-Effekt“ aus der Sommer-Ausgabe inspiriert. Die gibt es in dieser Ausgabe ebenfalls wieder. Diesmal kramen sie in der Tool-Kiste der Shell, um verschiedene Zahlensysteme ineinander umzuwandeln.

Die Fotografin des Titelbildes widmet eben dieses Armin K. Der klagte nämlich am 7. Dezember, dass der Morgen zu früh wäre und „dieses weiße Zeug da draußen“ das auch nicht besser mache. Zwar war der Morgen wirklich früh und kalt, bezeugt Hella Breitkopf, aber die Raureifnadeln waren nach einem nebligen Abend zuvor an manchen Stellen wirklich sehr dekorativ. Fotografiert hat sie das Titelbild mit einer *Olympus PEN-F*, 17-Millimeter-Objektiv und Blende 2 (und Lederhandschuhen).

Jobs zu vergeben!

Wir verzeichnen in dieser Ausgabe eine ungewöhnliche Häufung ganz bestimmter Art. Stellenausschreibungen sind in der GUUG ja nicht unbedingt etwas Tägliches. Dafür bietet diese Ausgabe gleich zwei davon. Beide Jobs sind Teilzeit, beide sind natürlich auf Honorarbasis (also keine Festanstellung), und beide sind sehr eng mit der GUUG verbunden.

Das eine ist die Assistenz für den Vorstand. Das andere ist eine neue Chefredaktion für die UpTimes! Die derzeitige Chefredaktion (meine Wenigkeit) muss wegen anderer beruflicher Verpflichtungen nach der Sommerausgabe 2017 die UpTimes in artikelverliebte Unix-Hände abgeben.

Wünschenswert ist, wenn sich die UpTimes-Chefredaktion als „just another geek“ begreift, um ein Gespür für die richtigen Themen zu haben. Mindestens jedoch ebenso wichtig ist, das redaktionelle Geschäft ernst zu nehmen und für das Redaktionsteam organisatorisch zusammen zu halten. Der Kasten „Stellenausschreibung: Chefredaktion für die UpTimes“ versucht zu umreißen, auf was es im Einzelnen ankommt.

Bei Fragen fragen. Und bei Verdacht auf Interesse außerhalb der GUUG-Mitglieder unbedingt weitersagen. Die jetzige Chefredaktion war auch nie GUUG-Mitglied und hat es trotzdem ganz passabel gemacht. Oder? Oder?!?

Über Anika

Anika Kehrer arbeitet seit zehn Jahren als Technikjournalistin und ist seit der Wiederauflage der UpTimes im Sommer 2012 von der GUUG als Chefredakteurin beauftragt. Ihre Arbeit ist auszugsweise auf der Webseite https://www.torial.com/anika.kehrer ersichtlich. Ihr Ziel für die UpTimes ist, sie als Vereinszeitschrift der GUUG in Zusammenarbeit mit GUUG-Mitgliedern als hochwertiges und partizipatives Fachmagazin zu gestalten.

Stellenausschreibung: Chefredaktion für die UpTimes

Die German Unix Users group (GUUG e.V.) sucht zum 1. Juli 2017 eine **Chefredaktion** für die *UpTimes*, die Mitgliederzeitschrift der GUUG.

Gesucht ist ein Mitglied oder eine externe Person, die sich der UpTimes, dem Mitgliedermagazin des GUUG e.V., gesamtverantwortlich annimmt. Sie koordiniert und produziert federführend zwei bis vier Ausgaben pro Jahr. Journalistische oder redaktionelle Erfahrung ist sehr nützlich, aber nicht zwingend notwendig. Eine grundsätzliche Affinität zu technischen Themen ist erwünscht, insbesondere rund um unixoide Betriebssysteme und damit verbundener Anwendungen. Zu den Aufgaben zählen:

(1) Artikelakquise und Autorenkorrespondenz: Suchen und Finden von Themen und Autoren.

(2) Textredaktion und Bilderrecherche: inhaltliche und sprachliche Qualitätssicherung sowie Sicherstellen, dass das Magazin ausreichend viele Abbildungen, Autorenbilder und Bildunterschriften enthält.

(3) Zusammenarbeit mit und Förderung der ehrenamtlichen Redaktionsmitglieder: Mailinglisten-Updates, Korrespondenz mit der Redaktion und Außenstehenden, Pflege der Organisationsseite für jede Ausgabe im GUUG-Wiki.

(4) Vorbereiten der UpTimes-Inhalte für ihre Verarbeitung durch die Produktions-Toolchain, die `git` und LaTeX enthält. Für den Satz und die Betreuung der IT-Systeme wird gesorgt.

(5) Weiterentwicklung des Layouts und der grafischen Gestaltung in Absprache mit den dafür zuständigen Mitgliedern.

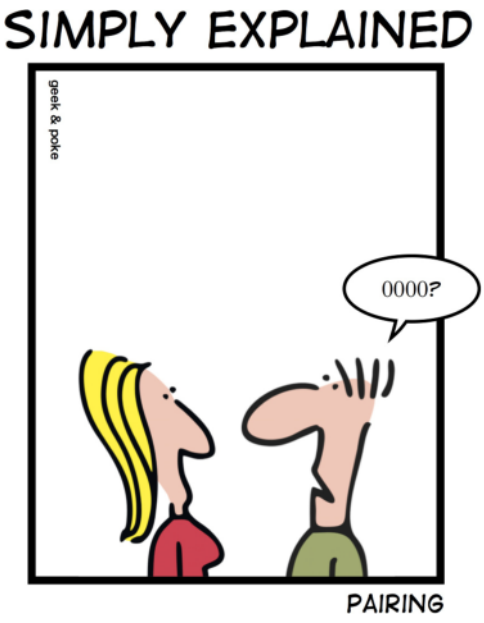
(6) Zusammenarbeit mit den Organen und Gruppen innerhalb der GUUG: Unterstützung von Anliegen des Vereines unter Wahrung der redaktionellen Unabhängigkeit.

(7) Handling der Autorenabrechnungen in Form von Honorarberechnung und Informationsverteilung an Autoren und Kassenwart. Die Rechnungen selbst schicken die Autoren an den Verein, der sie auch abwickelt.

Es sind verschiedene Vergütungsmodelle möglich, etwa auf Stundenbasis oder pauschal pro Ausgabe. Büroeinrichtung und Kommunikationsinfrastruktur organisiert die Chefredaktion selbständig. Die Wahl des Arbeitsortes ist frei wählbar. Der Vorstand vergütet Materialaufwände und Reisekosten nach Absprache, zum Beispiel die Teilnahme am FFG, der hauseigenen Fachkonferenz der GUUG, zu Berichts- und Kontaktzwecken.

Interessierte melden sich bitte zusammen mit einer Beschreibung ihrer Motivation und ihrer redaktionellen Erfahrung bei der Redaktion unter <redaktion@uptimes.de>, die für Fragen zur Verfügung steht.

Dieses Stellengesuch kann gern an geeignete Stellen innerhalb und außerhalb des Vereins weitergereicht werden.



/etc
Neues aus dem Verein

Die *GUUG* lebt von den Aktivitäten und Plänen ihrer Mitglieder. An dieser Stelle berichten wir von Neuigkeiten aus allen Bereichen des Vereins. Diesmal geht es um die neuen Vereinsarbeitsgruppen, die Vorstandswahl, das FFG 2017, die Fusion mit dem *LinuxTag*-Verein, die neue Benutzerordnung und das Stellengesuch für eine Vorstandsassistenz.

von **Nils Magnus**

Reden ist Silber,
handeln ist Gold,
gemeinsam handeln ist Dynamit.

Frei nach dem bekannten Sprichwort unbekannter Herkunft.

Verein definiert Arbeitsgruppen

Der Vorstand versucht, die inhaltliche Arbeit der GUUG besser zu strukturieren. Dafür hat er drei Vereinsbereiche definiert und jeweils einen Ansprechpartner für jeden Bereich abgestellt, der dort die Arbeit koordiniert. Damit die Bereiche jedoch auch Leben erhalten, sind sie auf Vorschläge, Rückfragen und Engagement der Mitglieder angewiesen.

Der Bereich *Veranstaltungen* (Kontakt: Nils Magnus) bündelt die zentralen und dezentralen Events. Der Bereich *Publikationen* (Kontakt: kommissarisch Felix Pfefferkorn) möchte den Wissensschatz heben und verteilen, der im Verein und bei seinen Mitgliedern liegt. Dazu gehört unter anderem die UpTimes oder die Idee, auch in anderen Fachmedien zu publizieren. Der dritte Bereich *Ausbildung* (Kontakt: Felix Pfefferkorn) fokussiert sich auf den Nachwuchs und die Förderung von Talenten aus dem Unix-Bereich.

Alle drei Bereiche haben eine Seite auf der GUUG-Homepage eingerichtet. Dort sind auch die fachlichen Themen gelistet, die die Bereiche orthogonal bearbeiten wollen.

Vorstandswahlen angekündigt

Im März 2017 endet die aktuelle Amtsperiode des Vorstands. Davor stehen also Neuwahlen an. Über den genauen Ablauf und die Termine informiert die von den Mitgliedern gewählte Wahlkommission, die ihrerseits Hanno Wagner als Wahlleiter bestimmt hat.

Es sind noch Plätze frei im Vorstands-Dreamteam 2017/18. Mitglieder, die Interesse an der Gestaltung unseres Vereins haben, wenden sich mit Fragen wahlweise an den Vorstand oder mit ihrer Kandidatur direkt an die Kommission.

Frühjahrsfachgespräch 2017

Im Status „orange“ befindet sich gegenwärtig die Planung des Frühjahrsfachgesprächs 2017. Obwohl es bereits Vorschläge und Anfragen zu einem Veranstaltungsort gibt, fehlen immer noch engagierte Mitglieder, die Ideen und Vorschläge zur GUUG-Hauskonferenz einbringen – insbesondere jemand, der sich den Hut der Gesamtkoordination aufsetzen möchte. Der Vorstand hilft gerne mit Rat, praktischer Unterstützung und weiteren Ressourcen.

Fusion mit *LinuxTag*

Wie auf der letzten Mitgliederversammlung diskutiert, hat der Vorstand Verhandlungen mit dem LinuxTag zu einer Fusion aufgenommen. Der Vorstand hat beschlossen, diese Pläne weiter zu verfeinern. Eine Arbeitsgruppe arbeitet zusammen mit einem Fachanwalt an einer so genannten „Verschmelzung durch Aufnahme“, bei der der LinuxTag der GUUG beitreten würde. Sobald die Informationen aufbereitet sind, werden die Details der Fusion allen Mitgliedern beider Vereine zugänglich gemacht. Wer Interesse hat, an diesem Prozess mitzuwirken, meldet sich bitte bei Nils Magnus im Vorstand.

Benutzerordnung verabschiedet

Um einerseits den Zugriff auf die von der GUUG betriebene IT auf solide Füße zu stellen und außerdem auch vereinsexternen Dienstleistern Zugriff auf Teilbereiche unserer Daten zu gewähren, hat der Vorstand nach Diskussion mit den aktiven Mitgliedern eine Benutzerordnung beschlossen. Sie gilt ab dem 1. Dezember 2016 für alle Nut-

zer und findet sich unter https://wiki.guug.de/orga/vorstand/benutzerordnung.

Vorstandsassistenz gesucht

Die operativen Tätigkeiten des Vorstandes erfordern regelmäßige Aufmerksamkeit, etwa das Führen von Protokollen, regelmäßige Korrespondenz, das Nachhalten von Aufgaben oder die Vorbereitung von Veranstaltungen. Daher hat der Vorstand beschlossen, eine Stelle als Vorstandsassistenz auszuschreiben, damit sich Vorstand und Mitglieder stärker auf strategische Tätigkeiten fokussieren können.

Der Vorstand präferiert eine stundenweise Vergütung dieser Aufgabe. Im Kasten „Stellenausschreibung: Vorstandsassistenz für die GUUG“ befindet sich eine Profil der Stelle. Wer jemanden für diese Position empfehlen kann, die sich gut für eine Teilzeitbeschäftigung eignet, oder wer selbst Interesse hat, wendet sich bitte an den Vorstand. Das Stellengesuch kann gern an geeignete Stellen innerhalb und außerhalb des Vereins gelangen.

Stellenausschreibung: Vorstandsassistenz für die GUUG

Die German Unix Users group (GUUG) sucht zum 1. Januar 2017 eine **Vorstandsassistenz** auf Stundenbasis.

Die Aufgabe der Vorstandsassistenz ist die organisatorische Durchführung operativer Vereinstätigkeiten. Hierzu zählen in Abstimmung mit dem Vorstand:

(1) Vor- und Nachbereitung der Mitgliederversammlungen,

(2) Vor- und Nachbereitung sowie Protokollierung der Vorstandssitzungen,

(3) Vorantreiben der von Mitgliederversammlung und Vorstand beschlossenen Aufgaben,

(4) Kontaktpflege zu (potentiellen) Mitgliedern, Sponsoren, Referenten und Multiplikatoren, sowie

(5) Zusammenarbeit mit den – ehrenamtlichen und bezahlten – Mitarbeitern des Vereins und nach Rücksprache mit dem Vorstand die Koordination ihrer Tätigkeiten.

Die Vergütung erfolgt auf Basis einen Stundensatzes. Für die abrechenbare Stundenzahl wird eine monatliche Obergrenze vereinbart. Nach Rücksprache mit dem Vorstand kann die Obergrenze erhöht werden. Die Abrechnung erfolgt monatlich.

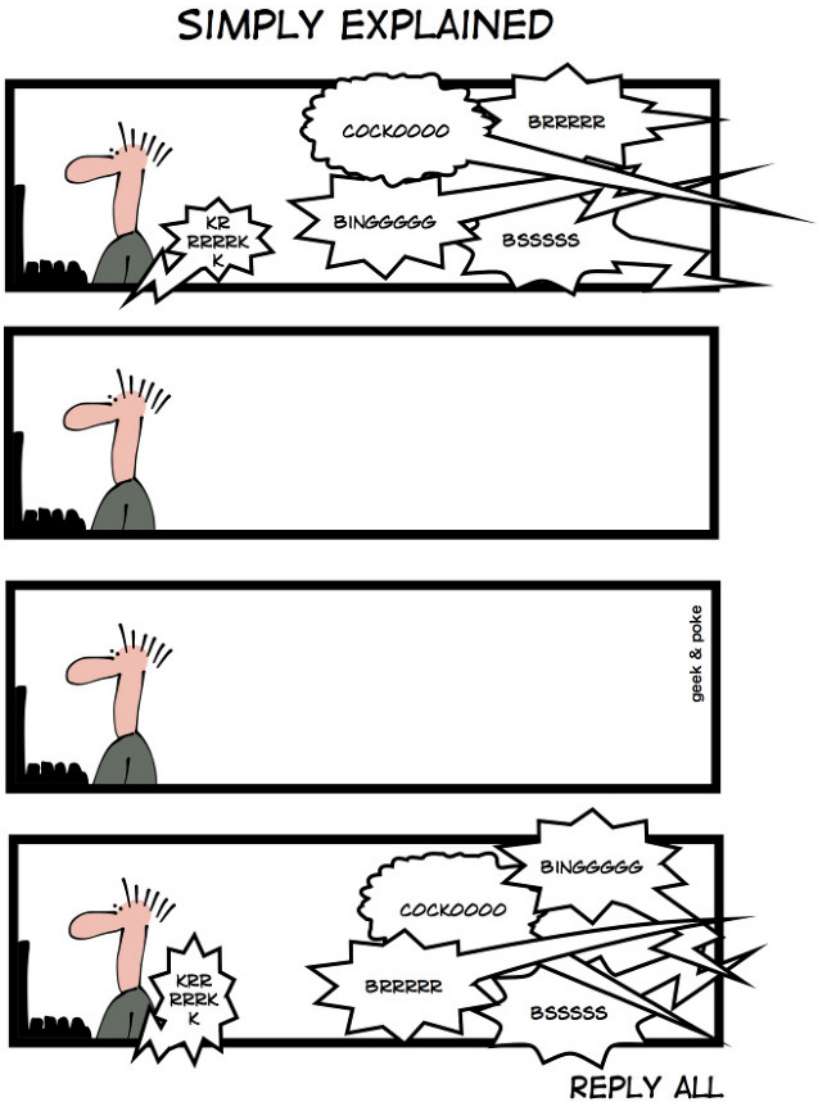
Die Vorstandsassistenz legt dem Vorstand im Vorfeld seiner Sitzungen, ggf. auch häufiger, einen schriftlichen Bericht vor. Büroeinrichtung und Kommunikationsinfrastruktur organisiert die Vorstandsassistenz selbständig. Die Wahl des Arbeitsortes ist frei wählbar. Der Vorstand vergütet Materialaufwände und Reisekosten nach Absprache.

Interessierte melden sich zusammen mit einer Selbstbeschreibung und Vergütungsvorstellung beim Vorstand unter <vorstand@guug.de>.

Dieses Stellengesuch kann gern an geeignete Stellen innerhalb und außerhalb des Vereins weitergereicht werden.

Über Nils

Nils Magnus ist seit 2014 Mitglied im Vorstand des *German Unix Users Group* e.V. (GUUG). Beruflich verdingt er sich als leitender Security-Consultant und Fachautor. Er befasst sich mit Vorliebe mit Cloud-Infrastrukturen auf Basis von Open Source. In seiner Rolle als Veranstalter beim *LinuxTag* und der GUUG organisiert er seit über 15 Jahren Konferenzen und Workshops. Nils lebt in München und streckt schon ein Bein nach Berlin aus.



Aufruf
GUUG-Vorstandswahl 2017

Die Wahlkommission ermutigt alle Mitglieder, für einen Posten im Vorstand zu kandidieren.

von **Hanno Wagner, Wahlleiter GUUG e.V.**

An ihren Früchtchen sollt Ihr sie erkennen.

Frei nach Matthäus 7,16

Liebe GUUG-Mitglieder,

zu Beginn des kommenden Jahres steht wieder eine Vorstandswahl an, die entsprechend unserer Satzung per Briefwahl durchgeführt wird. Zu wählen sind:

- ein Vorsitzender,
- zwei Stellvertreter,
- ein Schatzmeister,
- bis zu fünf Beisitzer.

Die Wahlkommission ermutigt Euch, für einen Posten im Vorstand zu kandidieren. Berechtigt hierzu ist jedes persönliche Mitglied. Informationen über Aufgaben und zeitlichen Aufwand der Vorstandsmitglieder erteilt der amtierende Vorstand gern.

Eine Kandidatur kann unter Angabe des angestrebten Amtes, des Namens und der Mitgliedsnummer bei der Wahlkommission eingereicht werden. Sie muss bis zum **14.01.2017** eingegangen sein. Der Eingang wird durch den Wahlleiter bestätigt.

Eine Einreichung ist möglich als Brief oder E-Mail an:

GUUG e.V. Wahlkommission
co Hanno Wagner
Mittelfeldstraße 23
70806 Kornwestheim
E-Mail: <kandidatur@guug.de>

Bis zum **12.02.2017** sollten die Kandidaten eine Kurzvorstellung (maximal 1500 Zeichen) mit Bild zur Beilage zu den Wahlunterlagen in elektronischer Form einreichen, ein Formatvorschlag wird zur Verfügung gestellt. Geplant ist auch ein Interview, in dem die Kandidaten zum Beispiel zu aktuellen Themen und Herausforderungen Stellung nehmen können.

Aus den gültigen Kandidaturen erstellt die Wahlkommission in Zusammenarbeit mit den Kandidaten die Unterlagen zur Briefwahl, die voraussichtlich am 20.02.2017 verschickt werden und dann bis zum 18.03.2017 an die Wahlkommission zurückgeschickt werden müssen. Die Bekanntgabe des Ergebnisses sowie die Amtsübergabe an den neuen Vorstand erfolgen dann voraussichtlich am 23.03.2017.

Satzung und Wahlordnung finden sich auf unseren Webseiten unter http://www.guug.de/verein/satzung/.

Mit freundlichen Grüßen, Hanno Wagner

Hamwa gut gemacht
Rückblick auf die *OpenPGP.conf*

Die GUUG hat im Herbst die erste *OpenPGP.conf* ausgerichtet. Eine kleine Umfrage unter den Teilnehmern hat erschlossen, wie das Ganze geglückt ist.

von **Nils Magnus**

Wieso, weshalb, warum
Wer nicht fragt bleibt dumm

Sesamstraße

Am 8. und 9. September 2016 fand im *Leonardo Royal Hotel Köln Stadtwald* die erste *OpenPGP.conf* statt. Gekommen waren etwas über 50 Teilnehmer, was Besucher, Sprecher (etwa 20) und Organisatoren (der verantwortliche Organisator Martin Schulte, dann Werner Koch, Erwin Hoffmann und Nils Magnus) einschließt.

An zwei Tagen diskutierten die Teilnehmer aus gut einem Dutzend Länder auf hohem fachlichem Niveau über Trends in der Verschlüsselungsszene. Die Teilnehmer stellten der Konferenz ein gutes Zeugnis aus in den Kategorien *Venue*, *Format* (es gab nur einen parallelen Track) sowie für die Themen *Kryptographie* und *GPG-Weiterentwicklung*. Darum steht eine Neuauflage im Raum.

Umfrage gemacht

Am zweiten Tag hat der Autor dieser Zeilen eine Umfrage zusammengestellt, die auf der Konferenz bekanntgegeben wurde und die auch als Grundlage für weitere Konferenzangebote dienen soll. Binnen einer Woche nahmen 20 Personen an der anonymen Umfrage teil (es war weder eine Registrierung nötig, noch wurden Name oder E-Mail-Adresse abgefragt). Obwohl es keinen Schutz gegen Doppelausfüllungen oder sonstige Manipulationen gab, gibt die manuelle Durchsicht der Antworten keinen Anlass, ihr Auftreten zu befürchten.

Auch wenn durch die geringe Grundgesamtheit die Umfrage zwangläufig nicht repräsentativ sein kann und sie auch nur die Meinungen der Teilnehmer erfasst – also nicht die derjenigen die sich bewusst oder unbewusst gegen die Konferenz entschieden haben – lassen sich einige Hinweise und Schlüsse aus den Antworten ziehen.

Die Teilnehmer sollten zumeist eine Aussage auf einer Skala von 1 bis 5 bewerten, wobei 5 die größte Zustimmung signalisierte.

Aussagen zum Konferenzformat

Das Format als Zwei-Tages-Konferenz erhielt mit einem Wert von 4,1 (von 5) hohe Zustimmung. Noch eine Spur stärker war die Zustimmung zu dem gewählten Single-Track-Format (4,2). Deutlich positiv bewerteten die Teilnehmer auch die Entscheidung, die Konferenz unter der Woche stattfinden zu lassen (3,7). Dazu passt, dass 39 Prozent während ihrer Arbeitszeit als Angestellte kamen, weitere 22 Prozent zeitlich flexible Freiberufler sind, und ein Drittel ihre eigene freie Zeit für die OpenPGP.conf aufwendeten.

Organisatorisch kam der Vorschlag, mehr Zeit für Working Sessions, Workshops, Debatten, Diskussionen oder Unconference-artige Sessions anzubieten. Andere lehnten allerdings genau das ab. 78 Prozent der Besucher bewerten die Idee eines Hackfests als gut oder sehr gut, wohingegen 22 Prozent damit gar nichts anfangen können.

Für eine zukünftige Ausgabe der OpenPGP.conf legen diese Antworten nahe, wieder eine zweitägige One-Track-Konferenz unter der Woche anzubieten, da damit die höchste Zustimmung zu erzielen ist, selbst wenn es nicht sämtliche Teilnehmer zufriedenstellt. Ein paar Optionen für freiere Formate oder gar ein weiterer Tag, der in ein Wochenende hineinreicht, wären vielleicht eine Ergänzung.

Aussagen zu Kosten und Angebot

Leichte Unzufriedenheit gab es bei den Kosten der Konferenz (Zustimmung 2,8), wobei sich kein klarer Trend herausbildet: Für zwei Drittel waren

die Kosten akzeptabel oder noch besser, aber immerhin ein Drittel war unzufrieden. Kein Wunder, denn 28 Prozent zahlten die Kosten aus eigener Tasche, während bei 39 Prozent der Beitrag vom Arbeitgeber kam. Das restliche Drittel hat keine Teilnahmegebühren bezahlt. Die Reisekosten trugen die Teilnehmer überwiegend selbst (58 Prozent), bei 39 Prozent zahlte der Arbeitgeber.

Mit dem Veranstaltungsort (inklusive Saal, Technik und Snacks) waren die Teilnehmer recht zufrieden (4,2), mit dem Hotel weitgehend zufrieden (3,5). Einige äußerten sich explizit positiv zum Ort und wollten sofort zurückkehren, andere bemängelten den Ort als zu luxuriös und zu wenig vegetarierfreundlich.

Aussagen zu Konferenzinhalten

Die Anwesenden befürworten leicht mehrheitlich die Idee einer Einführungssession, die an das Thema OpenPGP heranführt, sind aber auch nicht außergewöhnlich begeistert davon (3,6). Vermutlich läge dieser Wert bei Nicht-Teilnehmern höher.

Die Idee, das Programm stärker zu strukturieren, wird eher verhalten bewertet (2,8). Bemängelt wurde die späte Veröffentlichung des Programms – alles erschien den Umfrageteilnehmern etwas kurzfristig organisiert. Ein Teilnehmer wünschte sich auch mehr Informationen und Inhalte.

Die Umfrage hat sich nach Themenschwerpunkten erkundigt: Spitzenreiter waren kryptographische Themen mit 4,5 Punkten. Allgemeine OpenPGP-Themen schnitten mit 4,3 Punkten ab. Explizite GnuPG-Entwicklung erreichte 4,1 Punkte. Die Integration in MUAs bewerteten die Teilnehmer mit 3,8. Mit 3,2 Punkten lagen Hardware-Themen etwas zurück.

In den Kommentaren merkten Besucher an, dass es zwar viele Beiträge zur User Experience gäbe, es aber kaum User gäbe, um diese Aussagen zu belegen. Einige wünschten sich etwas mehr strategische Einordnung des Status Quo sowie der Roadmap von OpenPGP. Die Diskussion von Javascript-Implementierungen wurde vermisst.

Erfreuliche 78 Prozent der Konferenzbesucher können sich gut oder sehr gut vorstellen, selbst einen Beitrag für eine kommende Konferenz einzureichen. Über 94 Prozent empfehlen die OpenPGP.conf weiter. Sämtliche Umfrageteilnehmer können sich vorstellen, an der nächsten Ausgabe der OpenPGP.conf teilzunehmen.

Alles im Griff
Projektdokumentation mit *reStructuredText* und *Sphinx*

Dieser Artikel beschreibt häufige Missstände und Probleme bei der Dokumentation von Software-Projekten. Mit der textbasierten Markup-Sprache *reStructuredtText* und des Open-Source-Tools *Sphinx* zeigt er einen Lösungsansatz auf.

von **Lenz Grimmer**

Ordnung ist die Verbindung
des Vielen nach einer Regel.
Immanuel Kant

Um Entwicklern und Projektmanagern die Dokumentation eines Software-Produkts so einfach wie möglich zu machen, ist es wichtig, so viele Hürden wie möglich aus dem Weg zu räumen. Das Erstellen und Aktualisieren der Dokumentation muss integraler Bestandteil des Workflows werden. Wenn möglich, erfolgt es darum unter Einsatz bereits bekannter Werkzeuge.

Dieser Artikel beschreibt häufige Missstände und Probleme bei der Dokumentation von Software-Projekten. Anschließend zeigt er unter Zuhilfenahme des Open-Source-Tools *Sphinx* und der textbasierten Markup-Sprache *reStructuredt-Text* einen Lösungsansatz auf.

Ausgangssituation: Verwaiste Office-Dokumente

Als Ausgangssituation findet man in vielerorts Szenarien wie das Folgende. Mitunter gibt es bereits eine „Projektdokumentation“. Aber sie besteht oft aus mehreren Word- oder Libre-Office-Dokumenten auf einem zentralen Fileserver, oder befindet sich als Asset in einem Dokumentenmanagementsystem. Solche Dokumente wurden häufig irgendwann mal angelegt, etwa, weil ein neu hinzugekommener Teamleiter den Auftrag erteilte, mal Licht ins Projekt-Dickicht zu bringen. Daraufhin goss jemand den zu diesem Zeitpunkt aktuellen Stand des Projekts in ein Dokument, vergaß es und fasste es nie wieder an. Dokumentation in dieser Form wird erfahrungsgemäß selten bis gar nicht mit der laufenden Entwicklung synchronisiert. Es mangelt zum Beispiel an Zugriffsberechtigungen, oder auch schlicht an der Kenntnis über die Existenz eines solchen Dokuments.

Neben der so entstehenden mangelnden Aktualität haben Office-Dokumente in einem zentralen Dateiarchiv aber noch weitere Nachteile. Office-Dokumente machen Teamarbeit schwer bis unmöglich, insbesondere wenn mehr als ein Autor zeitgleich Änderungen an der Dokumentation durchführen soll. Auch wenn die gängigen Office-Anwendungen rudimentäre Unterstützung für Versionierung und Konsolidierung von Inhalten bieten, ist die korrekte Benutzung dieser Funktionalität den meisten Anwendern nicht geläufig. Letztendlich ist es doch wieder die Aufgabe eines Einzelnen, die verschiedenen Änderungen und Versionen zusammenzufassen und den letzten Stand in das Archiv hochzuladen.

Das macht insbesondere die Durchführung von Detailverbesserungen so unattraktiv, weil sie so aufwändig sind, dass sie schlichtweg nicht vorgenommen werden.

Auch die Dokumentierung verschiedener Software-Versionen ist bei Einsatz von Office-Dokumenten eine Herausforderung, also die parallele Dokumentation eines Projekts, das zum Beispiel in einer Release- und einer Entwicklungsversion existiert. Nun müssen nicht nur die Änderungen verschiedener Autoren berücksichtigt werden, sondern auch noch die Frage, in welche Versionszugehörigkeit des Dokuments sie zu übernehmen sind.

Anforderungen an ein Dokumentationsmanagementsystem

Um diese Defizite und Herausforderungen zu meistern, sollte ein optimales Dokumentationssystem folgenden Anforderungen entsprechen.

Der Quellcode der Dokumentation sollte im Plaintext-Format vorliegen, also als reine Textdateien. Dadurch ist die größtmögliche Plattformunabhängigkeit gewährleistet, da die Bearbeitung nicht nur über ein einzelnes Tool oder auf einer bestimmten Plattform erfolgen muss. Der Entwick-

ler kann stattdessen den Text-Editor seiner Wahl verwenden. Der Text lässt sich automatisiert mit gängigen Unix-Bordwerkzeugen wie *grep*, *sed* oder *awk* durchforsten und bearbeiten.

Reiner Text ermöglicht weiterhin eine einfachere Versionskontrolle und Kollaboration mit den vertrauten Entwicklerwerkzeugen und Methoden, also etwa Diffs, Branches und Merging. Damit ist die Dokumentation gefühlt näher am Code, was die Zugänglichkeit fördert. Es kommt also durch die Verwendung der gleichen Tools für Code-Entwicklung und Dokumentation zu keinem Medienbruch. Liegen Code und Dokumentation im gleichen Quelltext-Repository, lässt sich die Anpassung von Code inklusive der eventuell notwendigen Aktualisierung der Dokumentation im gleichen Commit einchecken. Das macht es deutlich einfacher, Code- und Doku-Änderungen im Nachhinein nachzuvollziehen.

Doch reiner Text allein reicht noch nicht aus. Notwendig ist auch eine Möglichkeit zur Formatierung und Strukturierung des Dokuments mit Hilfe textbasierter Markup-Elemente. Die Syntax sollte leicht zu erlernen sein, nicht zu sehr vom Inhalt ablenken und den Quelltext weiterhin lesbar erscheinen lassen. Hilfreich wäre auch die Unterstützung der Markup-Sprache durch den jeweiligen Text-Editor zum Beispiel durch Syntax-Highlighting, automatische Vervollständigung oder einfaches Anstoßen des Build-Prozesses.

Das Dokumentationsmanagementsystem sollte darüber hinaus automatisch verschiedene und möglichst plattformunabhängige Ausgabeformate generieren. Insbesondere sind hier PDF, HTML und ePub für E-Book-Reader zu nennen.

Idealerweise sollte die verwendete Software schließlich unter einer Open-Source-Lizenz verfügbar sein. Das bringt die bekannten Vorzüge mit sich: keine Lizenzgebühren, keine Abhängigkeit von einem bestimmten Hersteller, offene Formate, leichte Anpassbarkeit und Erweiterbarkeit.

Mögliche Kandidaten: Wikis

Basierend auf den formulierten Anforderungen kommen diverse Tools in Frage, die sich alle mehr oder weniger für Projekt-Dokumentationszwecke eignen. Häufig kommen bei der Projekt-Dokumentation zum Beispiel Wikis zum Einsatz. Diese erfüllen einige der genannten Kriterien und sind bereits ein deutlicher Fortschritt zur Ausgangssituation.

Wikis haben üblicherweise eine eigene Markup-Sprache, mit der sich Inhalte formatieren und strukturieren lassen. Durch die Implementierung als Webservice ist eine deutlich leichtere Teamarbeit möglich – jeder Wiki-Nutzer ist in der Lage, schnell und relativ komfortabel Änderungen an bestehenden Seiten vorzunehmen oder neue Kapitel zu erstellen. Die meisten Wikis bieten standardmäßig eine eingebaute Versionskontrolle, mit der sich die Historie eines Dokuments – „Wer hat wann was geändert?“ – nachvollziehen läßt. Auch die konkreten Differenzen zwischen zwei Dokument-Versionen können dargestellt werden.

Bei Wikis gibt es unzählige Optionen: neben dem Klassiker *MediaWiki* etwa auch *DokuWiki*, *MoinMoin* oder *Twiki*. Auch das proprietäre *Confluence*-Wiki von *Atlassian* ist sehr populär. Hier heißt es sehr genau abzuwägen, welche Funktionalität erfordert wird, wie sich das Wiki in bestehende Infrastruktur integriert und wie komfortabel man es bedient.

Während Wikis beim Thema Kollaboration punkten, haben sie jedoch einige systemimmanente Einschränkungen. Zum einen wäre zu nennen, dass sie neben der HTML-Ausgabe im Web-Browser meist kaum oder gar keine weiteren Ausgabeformate unterstützen. Der PDF-Export einer einzelnen Seite ist meist noch möglich. Aber wenn es darum geht, die Inhalte vieler Seiten als strukturiertes Dokument zu generieren, sind eher kreative Lösungen gefragt. Damit einher geht, dass die Erzeugung von Dokumentation mit einem Wiki häufig schlecht automatisierbar ist.

Weiterhin besteht immer noch ein Medienbruch zwischen Projekt-Code und Dokumentation. Der Entwickler muss kognitiv einen Kontextwechsel vollziehen, um nach der Veränderung des Codes die Dokumentation im Wiki nachzuziehen. Das erschwert insbesondere die Synchronisation zwischen Code und Dokumentation. Auch die Pflege mehrerer Versionen eines Dokuments erfordert manuelle Arbeit: Änderungen, die mehrere Versionen eines Projekts betreffen, müssen händisch per Cut & Paste in die entsprechenden Seiten oder Kapitel übernommen werden.

Markup und Buildchains

Um den Medienbruch zwischen Projekt-Code und Dokumentation zu vermeiden, bietet es sich an, auf Markup-Sprachen als Quelltext der Dokumentation zurückgreifen und diese als Text-Dateien zusammen mit dem Projekt-Code zu speichern.

Solche Quelltextformate versprechen, sich in unterschiedliche Ausgabeformate exportieren zu lassen, ähnlich wie ein Compiler aus Quellcode ein Executable für verschiedene Plattformen kompiliert.

Der Artikel von Patrick Koetter in der Sommer-Ausgabe der UpTimes [1] zeigt, welche Markup-Formate sich im Laufe der Zeit entwickelt haben, und stellt die Eigenschaften der geläufigsten Auszeichnungssprachen gegenüber. Zusätzlich zu den dort vorgestellten Markup-Dialekten wäre noch *Textile* erwähnenswert, das oft bei der Konvertierung in HTML-Format etwa bei Foren- oder Blog-Software zum Einsatz kommt. Für komplexere Dokumente eignet es sich allerdings weniger. Es gibt, ähnlich wie bei *Markdown*, über 20 verschiedene Dialekte und Implementierungen, die sich mehr oder weniger stark unterscheiden.

In der Praxis ist wichtig, wie und mit welchem Aufwand sich derart formatierte Quelltexte dann tatsächlich weiterverarbeiten lassen – wie also ihre Buildchain aussieht und welche Frameworks oder Werkzeuge es rund um das Markup-Format gibt. Während *DocBook XML* extrem leistungsfähig und ein de-facto-Standard für umfangreichere Dokumentationsprojekte ist, so ist es jedoch auch sehr schwergewichtig und erfordert eine steile Lernkurve. Das kann insbesondere unerfahrenene Entwickler abschrecken und damit wieder dazu führen, dass die Dokumentation schlicht nicht gepflegt wird.

Darüber hinaus ist der Quellcode eines DocBook-Dokuments im Original ohne spezielle Editor-Unterstützung nur schwer lesbar, da das Format sehr Markup-lastig ist und der Inhalt somit in den Hintergrund tritt. Auch der Build-Prozess von DocBook-Dokumenten mittels *FOP* ist durch das verwendete Java recht Ressourcen-intensiv und damit nicht ohne weiteres mal eben auf einer Entwickler-Maschine aufsetzbar. Ähnlich verhält es mit *AsciiDoc*. Während das Markup-Format an sich sehr schlank ist, benötigt es DocBook für die PDF-Ausgabe, was die Toolchain deutlich verlängert.

reStructuredText

Wegen der vielen Markup-Optionen und ihrer Ökosysteme ist es eine Geschmacks- und Glaubensfrage, für welche Lösung man sich entscheidet. Für den Autor ist das aus der Python-Welt stammende *reStructuredText* (*reST*, [2]) ein klarer Favorit.

Python-Projekte verwenden es neben der Dokumentation von Software-Modulen und darauf basierenden Applikationen auch für Kommentare im Quellcode. Ähnlich wie bei dem Tool *Doxygen* lassen die sich dann automatisiert extrahieren, um zum Beispiel eine API-Dokumentation für Entwickler zu erzeugen. Mit Hilfe des Tools *pydoc* ist das auch on-the-fly möglich. Für *reST* spricht weiterhin, das eine sehr aktive Entwicklergemeinde es weiterentwickelt, und dass es große Verbreitung auch außerhalb der Python-Community besitzt. Das *Docutils*-Projekt von Python ist Hüter der Markup-Sprache und bietet neben einer Python-Bibliothek zur Verarbeitung von *reST*-Dokumenten eine Reihe von Werkzeugen an, mit denen sich *reStructuredText* in die verschiedensten Ausgabeformate konvertieren lässt.

Ein in *reST* formatierter Dokument-Quelltext ist auch ohne speziellen Editor gut lesbar und selbst im Rohformat fast druckreif, wie das Beispiel in Abbildung 1 zeigt.

```
.. code:: rst

Chapter 1
==========

Hello world!

I hope *your day* is proceeding **splendidly**!

Chapter 2
==========

Hello again, world!
```

Abbildung 1: Quasi druckreif: Code-Beispiel für *reStructuredText*.

Rund um das Markup hat sich mittlerweile ein großes Ökosystem aus Anwendungen und Tools gebildet. So wandelt beispielsweise das Werkzeug *odt2rst* Dokumente aus Libre Office oder Openoffice.org in das *reST*-Format um. Die Ergebnisse hängen zwar stark von den im Originaldokument verwendeten Auszeichnungen und Formatierungen ab und müssen meist noch nachbearbeitet werden. Aber mitunter ist eine solche Konvertierung der schnellere Weg, als Inhalte per Cut & Paste aus einem Office-Dokument zu übertragen.

Auch die Erstellung von Präsentationen in *reST* geht mit Werkzeugen wie *Hovercraft* oder *Landslide* recht gut von der Hand. Die daraus generierten Folien sind dank der Javascript-Bibliothek *impress.js* (Hovercraft) und der Verwendung von HTML5 (Landslide) optisch sehr ansprechend. Die

Python-basierte Software *Nikola* erzeugt auf Basis von *reST*-Dokumenten Blogs oder ganze Websites, die dank ihrer statischen Natur sehr performant sind und sich quasi überall ohne viel Aufwand hosten lassen. *reStructuredText* hat sich also auch außerhalb des Kontexts der Projektdokumentation gut etabliert.

Sphinx

Um nun aber aus einzelnen Dateien mit *reST*-Markup eine ansprechende und strukturierte Dokumentation zu erzeugen, bedarf es eines darauf aufsetzenden Frameworks. Ein populärer Vertreter dieser Gattung ist *Sphinx* [3].

Es ermöglicht, auch komplexere Dokumente bestehend aus mehreren *reST*-Dateien zu strukturieren, zu verwalten und sie automatisiert in diversen Ausgabeformaten auszugeben. Es entstand ursprünglich für die Pflege und Verwaltung der Python-Dokumentation auf `http://docs.python.org/` und befindet sich weiterhin in aktiver Entwicklung. Von Haus aus unterstützt Sphinx die gängigen Ausgabeformate HTML, PDF und ePub. Die Erweiterung um andere Formate ist relativ einfach. Layout und Aussehen der fertigen Dokumente lässt sich mit Templates anpassen. Praktisch bei der HTML-Ausgabe ist die eingebaute Suchmaschine, die mit Javascript schnelles Durchsuchen eines Dokuments ermöglicht und auf Seiten des Webservers keinerlei Skriptsprachen oder andere Besonderheiten erfordert.

Sphinx erweitert die *reStructuredText*-Markup-Sprache um weitere Elemente, zum Beispiel das Einfügen von Querverweisen auf andere Dokumente oder automatisiert erzeugte hierarchische Strukturen wie Inhaltsverzeichnisse („toctrees“), Fußnoten und Glossare. Weitere *Sphinx*-spezifische *reST*-Konstrukte ermöglichen das Einfügen von dokumentationsspezifischen Elementen wie Info- und Warnboxen, Code-Beispiele mit farbigem Syntax-Highlighting und die automatische Substitution häufig vorkommender Text-Elemente wie Produktnamen.

Die umfangreiche und gut strukturierte Dokumentation bietet auch Anfängern einen leichten Einstieg. Bei Problemen findet sich schnell eine Antwort – Sphinx hat eine aktive und hilfreiche Community. Die Web-Plattform *ReadTheDocs* hostet öffentlich mit Sphinx generierte Dokumentationen aus der Community und bietet unter anderem die Möglichkeit, Querverweise auf andere dort gehostete Dokumente in die eigene Dokumentation einzubetten.

Wo viel Licht ist, ist natürlich auch Schatten. Es gibt auch bei *Sphinx* das ein oder andere „Gotcha!“, das einem Autor Kopfzerbrechen bereitet. Zu den Klassikern zählt sicher die gemischte Verwendung von Tabs und Leerzeichen im *reST*-Quellcode. Ähnlich wie bei Python dienen an vielen Stellen Einrückungen dazu, den Inhalt zu strukturieren. Wenn dafür nicht konsequent die gleichen Steuerzeichen verwendet werden, ist das Resultat mitunter überraschend, obwohl es im Quelltext auf den ersten Blick korrekt aussieht. Dies ist insbesondere bei Projekten mit vielen Entwicklern – und damit vielen verschiednen Text-Editoren – eine Herausforderung.

Auch die Substitution von Textelementen klappt nicht in allen Fällen. Probleme gibt es manchmal bei den sogenannten *literal blocks*, also Textblöcke, deren Inhalt ohne Formatierung konvertiert werden soll. Auch, was die Zeilenlänge angeht, sollte man die Ausgabe dieser Literal Blocks genau prüfen, da die Konvertierung zu lange Zeilen unter Umständen einfach abschneidet. Bei Code-Listings oder Kommandozeilenbeispielen sorgt das für Verwirrung beim Leser. Schließlich ist die Erzeugung von Tabellen, die am Ende sowohl in HTML und PDF schön aussehen, eine Herausforderung, die mit viel Versuch und Irrtum verbunden ist.

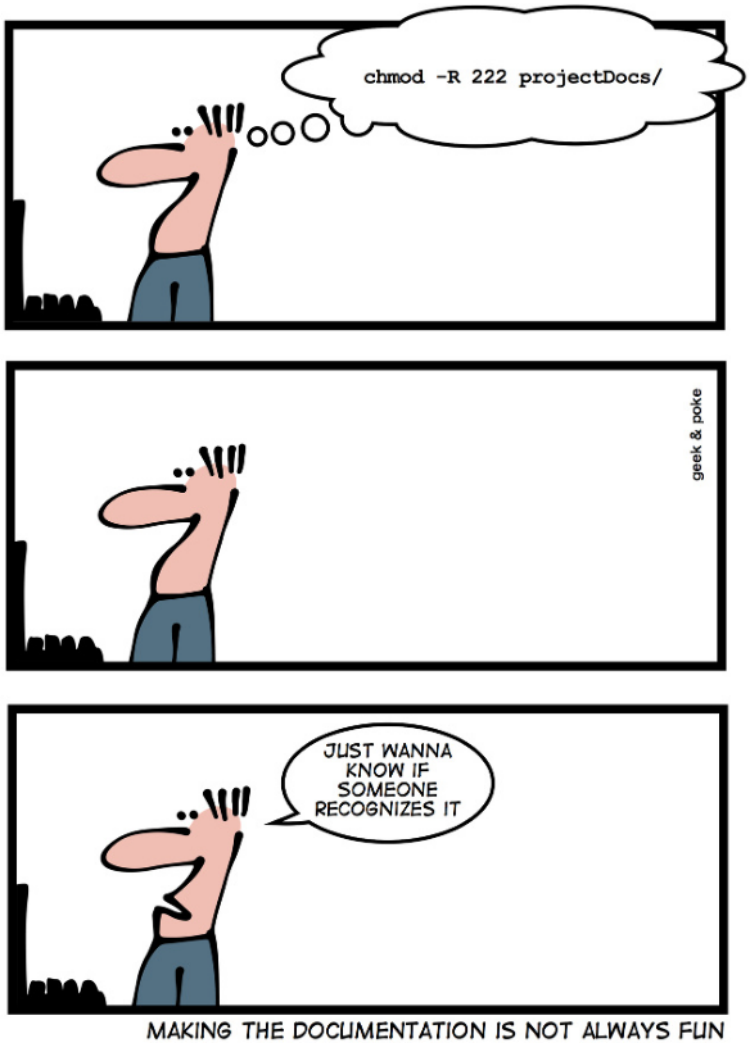
Der Autor findet: Die Gesamtzahl der Vorzüge macht diese Unannehmlichkeiten allemal wett.

Links

[1] Patrick Koetter: Was lange währt... Markup für Entwicklungsdokumentation. In: UpTimes 2016-1, S. 29: https://www.guug.de/uptimes/2016-1/
[2] *reStructuredText*: http://docutils.sourceforge.net/rst.html
[3] *Sphinx*: http://sphinx-doc.org/

Über Lenz

Lenz Grimmer ist bei der *SUSE LINUX GmbH* für die Weiterentwicklung des quelloffenen Storage-Managementsystems *openATTIC* verantwortlich. Er befasst sich seit seinem Informatikstudium in den 1990er Jahren privat und beruflich mit Linux und Open-Source-Software. Häufig ist er auf Veranstaltungen anzutreffen, wo er Vorträge zu den unterschiedlichsten OSS-Technologien hält (zum Beispiel Storage und Infrastruktur). Sein privates Blog und weitere Links sind unter http://blog.lenzg.net/ zu finden.



Erst du, dann du, dann du, dann du
Synchronisation von Jobs in der Shell

Nachdem die letzte UpTimes das Thema Pipes ausführlich für die Kommunikation erläutert hat, geht dieser Artikel auf einen anderen Aspekt ein, bei dem sich Pipes bewährt haben: die Synchronisation von Prozessen.

von **Mathias Weidner**

Skripten bringt alles in Ordnung.

Frei nach dem afrikanischen Sprichwort „Weinen bringt nichts in Ordnung."

Vor einiger Zeit kam ein Kollege auf mich zu und fragte, wie er am besten mehrere Backup-Jobs gleichzeitig starten könne. Außerdem sollten ein paar andere Jobs später laufen, wenn die ersten fertig sind. Das Ganze wollte er möglichst schnell und ohne zusätzliche Programme erledigen – also in der Shell.

Abbildung 1 veranschaulicht das Problem: Zum Zeitpunkt *t0* starten die ersten Jobs. Sie sind zum Zeitpunkt *t1* fertig. Die nächsten Jobs sollen zum Zeitpunkt *t2* starten. Eine dritte Staffel sollte frühestens zum Zeitpunkt *t3* starten, wenn alle Jobs der zweiten Staffel fertig sind. Außerdem sollen *t1* und *t2* nicht allzu weit auseinander liegen – es ging ja um Backup-Jobs, die über Nacht laufen und am Morgen fertig sein sollen.

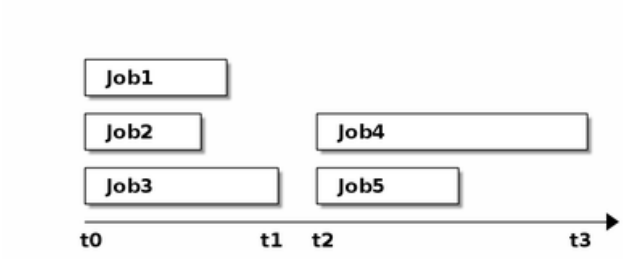


Abbildung 1: Mehrere Jobs in der Shell.

Die erste Idee, die Jobs mit `&` in den Hintergrund zu schicken, bringt einige Probleme mit sich:

- Bei Hintergrund-Jobs bekommt ein Shell-Skript nicht mit, wann sie fertig sind. Man muss *t1* und *t3* daher schätzen und verschenkt Zeit, weil zuviel Reserve bleibt oder die nächste Staffel zu früh startet, während die vorherige noch läuft.
- Die Ausgaben der einzelnen Jobs vermischen sich. Man kann nicht vorhersagen, wessen Ausgabe an welcher Stelle erscheint.
- Man bekommt keine Rückgabewerte der Jobs und kann somit im Skript nicht darauf reagieren.

Für diese Probleme gibt es Abhilfen, auch mit den Mitteln der Shell selbst. Doch die meisten benötigen Interprozesskommunikation und temporäre Dateien. Außerdem blähen sie das Skript auf, bis am Ende die eigentlichen Jobs kaum noch zu finden sind. Das möchte man nicht an einen Kollegen weitergeben.

Synchronisation mit Pipes

Es gibt allerdings eine einfache Möglichkeit, Jobs in der Shell gleichzeitig zu starten, bei der die Shell automatisch wartet, bis der letzte der Jobs geendet hat: Prozesse, die über Pipes miteinander verbunden sind, startet die Shell gleichzeitig.

Die Standardausgabe jeder dieser Jobs ist mit der Standardeingabe des nächsten verbunden, und die Standardeingabe mit der Standardausgabe des vorigen Jobs. Auf diese Art reichen die Ausgaben aber nur zum jeweils nächsten Job, und die Shell sieht nur die Ausgabe des jeweils letzten. Die Jobs davor würden zudem blockieren, wenn ihre Ausgabe nicht gelesen wird.

Nun ließe sich die Ausgabe der Jobs mit `cat` in eine Datei leiten und diese so entsperren:

```
( cat >> $logfile &; jobx )
```

Damit würden zwar alle Jobs problemlos laufen. Aber die Ausgabe würde wieder durcheinander in die Logdatei geschrieben. Besser ist es, wenn jeder Job die Ausgabe des vorigen Jobs an den nächsten weiterleiten würde. Das heißt, jeder Job, mit Ausnahme des ersten, müsste etwa so aussehen:

```
( jobx; cat )
```

Die runden Klammern schicken die Jobs in eine Subshell, sodass das Semikolon nicht die Pipe auseinanderreißt. Auf diese Weise erscheinen die Ausgaben allerdings in umgekehrter Reihenfolge:

```
$ echo 1|(echo 2; cat)|(echo 3; cat)
3
2
1
```

Außerdem kann Job Nummer 2 seine Ausgabe erst vollständig schreiben, wenn Job 3 beendet ist und `cat` die Ausgabe weiterleitet. Dito für Job 1: Zwar starten alle Jobs zur gleichen Zeit, aber die ersten blockieren, bis die letzten fertig sind. Das lässt sich zwar lösen, wenn die Reihenfolge des `cat`-Befehls und des Jobs getauscht ist:

```
$ echo 1|(cat; echo 2)|(cat; echo 3)
1
2
3
```

Nun stimmt zwar die Reihenfolge. Aber Job 2 startet erst, wenn Job 1 seine Ausgabe schließt. Und Job 3 startet erst, wenn Job 2 seine Standardausgabe schließt.

Hier hilft der Trick, die Ausgabe jedes Jobs in Variablen zwischenzuspeichern und sie erst am Ende auszugeben. Abgesehen davon, dass alle Jobs unabhängig voneinander laufen können, lässt sich die Ausgabe nun auch in die richtige Reihenfolge bringen. Damit es übersichtlicher wird, kapselt eine Shell-Funktion den Aufruf von `cat` und den Job, der ich den Job als Argument übergebe:

```
01 in_pipe() {
02   CMD="$*"
03   BEFORE=$(date)
04   LOG=$($CMD)
05   RESULT=$?
06   AFTER=$(date)
07
08   cat
09   echo $BEFORE
10   echo "$CMD"
11   echo "-----"
12   echo "$LOG"
13   echo "-----"
14   echo "RESULT=$RESULT"
15   echo $AFTER
16   echo "====="
17   return $RESULT
18 }
```

Diese Funktion zieht alle Register. Zeile 02 speichert alle übergebenen Argumente in einer Variablen, die als Befehl für den Job dient. Die Zeilen 03 und 06 notieren die Zeit vor und nach dem Aufruf des Jobs in zwei weiteren Variablen. Zeile 04 speichert die Ausgaben des Jobs in einer Shell-Variablen, während er läuft, und Zeile 05 hebt den Rückgabewert für später auf. Zeile 08 enthält den Aufruf von `cat`, der die Ausgabe des vorherigen Jobs durchleitet: Dadurch entspricht die Reihenfolge der Ausgaben der Reihenfolge der Jobs in der Pipe. Ab Zeile 09 kommt die Ausgabe aller gesammelten Werte des aktuellen Jobs, gefolgt von einer Trennzeile. In Zeile 17 schließlich gibt die Funktion den Rückgabewert des Jobs an die aufrufende Shell zurück.

Diese Funktion findet nun wie folgt Einsatz:

```
01 echo "=====" \
     | in_pipe sleep 3 \
     | in_pipe echo Hallo \
     | in_pipe sleep 2
02 date
```

Die Ausgabe sieht folgendermaßen aus:

```
=====
Mi 21. Jan 10:01:58 CET 2015
sleep 3
-----

-----
RESULT=0
Mi 21. Jan 10:02:01 CET 2015
=====
Mi 21. Jan 10:01:58 CET 2015
echo Hallo
-----
Hallo
-----
RESULT=0
Mi 21. Jan 10:01:58 CET 2015
=====
Mi 21. Jan 10:01:58 CET 2015
sleep 2
-----

-----
RESULT=0
Mi 21. Jan 10:02:00 CET 2015
=====
Mi 21. Jan 10:02:01 CET 2015
```

Alle Jobs starten also zur gleichen Zeit, werden aber zu unterschiedlichen Zeiten fertig. Das Skript fährt fort, nachdem der langsamste Job endete.

Probleme und Einschränkungen

Ein Problem gibt es noch, das mit folgendem Aufruf deutlich wird:

```
01 echo "=====" \
    | in_pipe false \
    | in_pipe sleep 1
02 echo $?
```

Das Problem wird in der letzten Zeile der Ausgabe offenbar:

```
=====
Mi 21. Jan 10:09:08 CET 2015
false
-----

-----
RESULT=1
Mi 21. Jan 10:09:08 CET 2015
=====
Mi 21. Jan 10:09:08 CET 2015
sleep 1
-----

-----
RESULT=0
Mi 21. Jan 10:09:09 CET 2015
=====
0
```

Die Ausgabe zeigt: Obwohl ein Job einen Fehlerwert gemeldet hat, registriert die Shell diesen nicht, weil der letzte Job in der Pipe ohne Fehler endete.

Bei der *POSIX*- oder *Bourne*-Shell muss man mit diesem Umstand leben – oder die Ausgabe der Pipe weiter untersuchen. Verwendet man jedoch *bash*, *ksh* oder *zsh*, erscheint folgende Lösungsanweisung im Skript vor der Pipe:

```
set -o pipefail
```

Allerdings hat auch diese Lösung Einschränkungen.

- Pro Job lässt sich nur ein Befehl aufrufen. Immerhin kann man Argumente mitgeben.
- Da die Ausgabe der Jobs in Variablen zwischenlagert, sollte die Ausgabe sich auf den maximalen Wert für Shell-Variablen beschränken. Wer mehr braucht, muss mit temporären Dateien arbeiten.
- Keine Ausgabeumleitungen lassen sich an die Funktion durchreichen. Das könnte man allerdings in die Funktion selbst einbauen, wodurch sie für alle Jobs gelten würde.

Wer das im Hinterkopf behält, kann verschiedene Jobs gleichzeitig aufrufen, ohne dass sich aufeinanderfolgende Staffeln von Jobs überlappen oder zwischendurch Zeit zu verschenken.

Über Mathias

Mathias Weidner studierte Ende der 1980-er Jahre Automatisierungstechnik in Leipzig. Nach verschiedenen Stellen in der Software-Entwicklung arbeitet er als Administrator für Unix/Linux, Netzwerke und Sicherheitsthemen. Seit einiger Zeit schreibt er hin und wieder Bücher dazu, die unter http://buecher.mamawe.net zu finden sind.

Läuft bei dir?
Betriebliche IT-Dokumentation in der Praxis (II)

Wie bekommen IT-ler die Herausforderung in den Griff, ihren Betrieb zu dokumentieren? Eine kleine Umfrage der *UpTimes* trat an, das herauszufinden. Teil 1 der Auswertung berichtete über das „Wie“. Teil II fragt hier nun nach dem „Was“.

von **Anika Kehrer**

Lexika sollen witzig sein!

Friedrich von Schlegel

Der erste Teil der Auswertung [1] zeigte, mit welchen Methoden die teilnehmenden IT-ler ihr Dokumentation bewältigen und wie zufrieden sie insgesamt sind. Wenn eine Bemerkung gestattet ist: Die Auswertung der Datensätze macht – gesetzt den Fall, man jongliert gern mit Zahlen – sehr viel Spaß und gestattet vielerlei Korrelationen. Es macht sich jedoch auch ausgesprochen bemerkbar, wie zeitaufwändig die Auswertung ist, wenn man mit gebührender Sorgfalt vorgehen möchte.

Es ließen sich also noch weitere Ergebnisse herausziehen, als dieser zweite Teil präsentiert. Dass das in dieser Ausgabe nicht geschieht, ist ein wenig schade. Andererseits: Die Datensätze laufen nicht weg. Und mögen sie noch mehr werden! Doch dazu später.

1 Stunde pro Woche

Unter den 69 Umfrageteilnehmern, die sich zur Dauer ihres Doku-Aufwandes geäußert haben, gibt es fünf Personen, die pro Tag mehr als eine Stunde schreiben. Es dürfte davon auszugehen sein, dass das dedizierte Doku-Zuständige sind. Denn die meisten der an der Umfrage teilnehmenden IT-ler dokumentieren ein Mal die Woche bis zu einer Stunde etwas. So verfährt jeder Zweite bis Dritte. Weiterhin wendet aber immerhin jeder Vierte tatsächlich jeden Tag bis zu einer Stunde Zeit auf, um etwas aufzuschreiben.

Betrachtet man nur, wie oft die Umfrageteilnehmer etwas dokumentieren, ohne dies in Zusammenhang mit der Dauer des Dokumentierens zu bringen, dann führen die Wöchentlich-Aufschreiber mit 31 von 67 Köpfen. Interessanterweise gibt es auf dem zweiten Platz aber mehr Personen, die täglich etwas aufschreiben, als Personen, die sich lediglich ein mal pro Monat um Dokumentation bemühen. Dieses Rennen ist mit 17 gegen 13 Nasen aber relativ knapp.

Lediglich drei Personen geben an, „Nie“ zu dokumentieren. Eine gute Handvoll berichtet, nur bei Bedarf etwas aufzuschreiben. Diese Angabe korreliert am häufigsten mit der Dauer „Bis 1 Stunde pro Monat“. Bei so kleiner Datenbasis ist eine Aussage über Korrelationen natürlich waghalsig. Aber zumindest so weit scheint es so, als ob bei „bedarfsorientierter“ Dokumentation der Bedarf als äußerst gering eingeschätzt wird.

Systeme: „Joa“, Abläufe: „Najaa“

Der Wust des so Erzeugten deckt relativ viel ab. Als Problem erscheint allerdings, dass die Hälfte davon veraltet und unvollständig ist. Bei satten drei Vierteln der Befragten (54 von 71 Personen) sind Sicherheitskonzepte niedergelegt, bei immerhin zwei Dritteln (42 von 71 Personen) Notfallpläne. Andererseits weiß jeder Sechste bis Siebte (15 Prozent, 11 Personen) nicht, ob es überhaupt so etwas wie einen Notfallplan gibt.

Mehrere merkten an, dass ein Dokument nicht unbedingt praktisch relevant ist, nur weil „Sicherheitskonzept“ oder „Notfallplan“ darauf steht. Die Beobachtung von formalistischer Dokumentenproduktion, nur mit dem Zweck, ein solches Dokument da zu haben, legt den Finger in die größte Wunde der IT-Betriebsdokumentation: Ihre Praxisnähe und Nützlichkeit.

Am leichtesten tun sich alle mit Netzwerkplänen. Logische Netze sind bei 75 Prozent der Befragten dokumentiert, das physische Netz bei 65 Prozent. Lückenhaft werden hingegen Operations-Dokumente für einzelne Dienste wahrgenommen: Da sind es 65 Prozent (46 von 71 Personen), die nur teilweise darauf zugreifen können, und knappen zehn Prozent (8 Personen) fehlen sie komplett.

Bei User-Anleitungen oder Support-Dokumente gibt es zwei Lager: Zwei Drittel haben sie (45 Teilnehmer), ein Viertel (17 Teilnehmer) hat so etwas aber überhaupt nicht. Jeder Fünfte kann nicht sagen, wie die Strom- und Glasfaserkabel laufen, oder ob das überhaupt jemand weiß.

71 Praxis-Snapshots!

Ein Dutzend neuer Datensätze haben sich seit dem ersten Teil der Auswertung ergeben. Haben sie das Gewicht der teilnehmenden Gruppen verschoben?

Bislang waren die allermeisten der teilnehmenden IT-Mitarbeiter sehr erfahren: Über 80 Prozent arbeiten seit mehr als zehn Jahren im IT-Bereich. Dieses Gewicht hat sich verstärkt, vor allem bei denjenigen, die sogar mehr als 16 Erfahrungsjahre zurückblicken. Die Altersgruppen haben sich im Bereich der 40- bis 50-Jährigen etwas verstärkt, ebenso die 30-bis-40-Jährigen auf dem zweiten Platz und die 50-bis-60-Jährigen auf dem dritten Platz.

Allerdings hat sich eine sechste Person gefunden, die die Gruppe der 20- bis 30-Jährigen überhaupt erst in die Statistik hebt. Gruppen unter sechs Personen gelten in dieser Auswertung als zu schwach, um gezählt zu werden. Ja, das ist ein vollkommen willkürlicher Wert. Aber irgendwo muss man die Grenze ziehen.

Dieselbe Neuerung hat sich für Unternehmensgrößen mit 11 bis 20 Mitarbeitern ergeben. Auch sie sind auf sechs Teilnahmen angestiegen und somit ab sofort Teil der Statistik. Hugh. Diejenigen, deren Betriebe mehr als 1000 Mitarbeiter haben, sind nach wie vor mit einem knappen Drittel aller Umfrageteilnehmer am stärksten repräsentiert. Am schwächsten sind diejenigen repräsentiert, deren Unternehmen weniger als 50 Mitarbeiter haben.

So geht es weiter

Mit Stand von Dezember 2016 liegen 71 Datensätze vor. Es scheint erstrebenswert, die Datensätze weiter zu entwickeln. Die Umfrage bleibt daher online. Da die Aussagekraft der Ergebnisse mit der Größe der Datenbasis steigt, ermuntert Kasten „Vergrößerung der Datenbasis“ weiterhin zur Streuung des Umfragelinks [2] unter Fachkollegen, Bekannten und anderen Stakeholdern zum Thema der betrieblichen IT-Dokumentation.

Vergrößerung der Datenbasis

Um die Datenbasis für die Erhebung der praxisnahen betrieblichen IT-Dokumentation zu erhöhen, sind mehr auswertbare Datensätze nötig. Das erste Rollout erfolgte binnen drei Wochen im Herbst 2015 und schöpfte aus zwei Kanälen: der *Google+*-Blase rund um den GUUG-Account, *Twitter* und die Mailingliste *SAGE@guug.de* mit rund 400 Subskribenden.

In ihrem ersten Rollout verzeichnete die Umfrage 56 auswertbare Datensätze. Mit Stand von Dezember 2016 liegen 71 Datensätze vor. Um das Bild nicht zu verzerren, sind Einzelgruppen mit weniger als sechs Repräsentanten in manchen Fragen kein Teil der Auswertung. Derzeit sind folgende Gruppen unterrepräsentiert: Jüngere zwischen 20 und 30 Jahren, Teamgrößen zwischen 15 und 25 Mitgliedern, Arbeitnehmer in Betrieben mit weniger als 50 Mitarbeitern.

Die UpTimes-Redaktion beabsichtigt, die Umfrage online zu lassen, damit sich im Laufe der Zeit die Datenbasis vergrößert. Wenn beispielsweise die Zahl 50 von neuen Datensätzen erreicht ist, erfolgt eine Aktualisierung der Auswertung.

Wer diese empirische Erhebung zu Lehr- und Lernzwecken für bessere betriebliche Dokumentation gut findet, streut sie bitte im deutschsprachigen Raum (die Umfrage steht aus Gründen der Machbarkeit nur in Deutsch zur Verfügung). Die Umfrage ist unter [2] zu finden.

Links

[1] Anika Kehrer: Weil es so schön ist; Betriebliche IT-Dokumentation in der Praxis (I). In: UpTimes 2015-2, S. 27: https://www.guug.de/uptimes/2015-2/
[2] Umfrage *Betriebliche IT-Dokumentation in der Praxis*: http://goo.gl/forms/9E6ev2ICVi

Shellskripte mit Aha-Effekt VII
Zahlenspiele

Manchmal muss man Zahlen zu einer anderen Zahlensystem-Basis darstellen, zum Beispiel vom Dual- oder Binärsystem (Basis 2) zum Sedezimal- oder Hexadezimalsystem (Basis 16). So lässt sich auch mit IP-Adressen oder Netzmasken herumrechnen.

von **Jürgen Plate**

Eure Rede aber sei: Ja! Ja! Nein! Nein!
Was darüber ist, das ist vom Übel.

Matthäus 5,37

Es ist immer wieder faszinierend, wie viel man mit der Shell erledigen kann, bevor man zu einer Programmiersprache greifen muss. Nicht zuletzt sind daran auch die zahlreichen Unterprogramme beteiligt, nämlich alle Kommandozeilen-Tools, die sich auf der Platte tummeln. Manchmal trifft man auch auf Systeme, die aus Sicherheitsgründen nur wenig mit Tools bevölkert sind. Aber auch da muss man nicht gleich aufgeben.

Bei den meisten Linux-Distributionen und Unix-Systemen ist der *bc* verfügbar. Das Tool *bc – An arbitrary precision calculator language* ist ein Rechner auf der Kommandozeile. Die GNU-Version ist sogar programmierbar. Er rechnet in beliebiger Genauigkeit und beherrscht auch etliche mathematische Funktionen, die der Kommandozeilenparameter `-l` aktiviert: Unter anderem sind das Sinus `s(x)`, Cosinus `c(x)`, beide im Bogenmaß), Arcustangens `a(x)`, die Exponentialfunktion `e(x)`, der natürliche Logarithmus `l(x)` und die Quadratwurzel `sqrt(x)`.

Der *bc* besitzt weder einen Eingabe- noch einen Ausgabeprompt. Nach der Begrüßung rechnet man im interaktiven Modus sofort los. Wie in der Shell kann man mit den Auf- und Ab-Pfeiltasten die letzten Eingaben zurückholen und editieren. Die Eingabe `quit` beendet das Programm im interaktiven Modus. In einer Pipe erledigt der *bc* seine Rechenaufgabe und beendet sich dann sofort.

Das folgende Listing zeigt, was passiert, wenn man einfach mal drauf los probiert (Zeile 03). Es gehen auch große Zahlen, wie Zeile 06 zeigt. Zeile 09 berechnet die Kreiszahl Pi: Der Standard sind 20 Stellen, aber wer mehr will, bekommt auch mehr. Zeilen 14 und 15 definieren eine Funktion und wenden sie an. Auch Gleitpunktzahlen gehen (Zeile 18). Am Schluss haben wir das übliche Problem bei begrenzter Stellenzahl:

```
01 $ bc -l -q
02
03 4*5+12+4
04 36
05
06 2^100
07 1267650600228229401496703205376
08
09 4*a(1)
10 3.14159265358979323844
11 scale=100; 4*a(1)
12 3.14159265358979323846264338327950288   \
        84197169399375105820974944592307   \
        81640628620899862803482534211706 76
13
14 define f(x) {x^3}
15 f(5)
16 125
17
18 2.345*2
19 4.690
20
21 (100/3)*3
22 99.9999999999999999
```

Aber dies soll ja kein *bc*-Kurs werden, sondern das Tool soll für Zahlenumwandlung zum Einsatz kommen.

Zahlenumwandlung mit *bc*

Standardmäßig rechnet der *bc*, wie auch andere Programme, im Dezimalsystem. Die Befehle `ibase=xxx` und `obase=yyy` legen die Basis des Zahlensystems getrennt für Eingabe- und Ausgabewerte fest. Die Umwandlung von beispielsweise einer Dezimalzahl in einen Binärwert erfolgt mittels `obase=2`. Als Pipe in einem Shell-Dreizeiler formuliert, sieht das folgendermaßen aus – die Ausgabe lautet dann wie gewünscht `Dez->Bin: 1111011`:

```
01 decimal=123
02 binary=$(echo "obase=2;$decimal" | bc)
03 echo "Dez->Bin: $binary"
```

Auf die gleiche Art erfolgt nun die Umwandlung der anderen Zahlensysteme.

`obase=10;ibase=2;` konvertiert binär nach dezimal, aber `ibase=2;obase=10;` konvertiert binär nach binär. Es empfiehlt sich, darauf zu achten, `obase` vor `ibase` zu modifizieren, weil die Modifikationen sofort aktiv werden und unerwartete Ergebnisse liefern können.

In dem folgenden Listing hatten wie die Pipe in Zeile 03 schon oben. Zeile 06 gibt als *Output Base* 16 statt 2 an, und schon hat man das Ergebnis als Sedezimalzahl. In Zeile 09 kommt die *Input Base*, mit deren Hilfe sich eine Binärzahl nach Hex konvertieren läßt. Die Zeilen 12 und 15 konvertieren die Ergebnisse der Zeilen 02 und 06 wieder zurück:

```
01 decimal=123
02
03 binary=$(echo "obase=2;$decimal" | bc)
04 echo "Dez->Bin: $binary"
05
06 hex=$(echo "obase=16;$decimal" | bc)
07 echo "Dez->Hex: $hex"
08
09 hex=$(echo "obase=16;ibase=2;$binary" | bc)
10 echo "Bin->Hex: $hex"
11
12 decimal=$(echo "obase=10;ibase=2;$binary" | bc)
13 echo "Bin->Dez: $decimal"
14
15 decimal=$(echo "obase=10;ibase=16;$hex" | bc)
16 echo "Hex->Dez: $decimal"
17 echo ""
```

Das Skript ergibt folgende Ausgabe:

```
Dez->Bin: 1111011
Dez->Hex: 7B
Bin->Hex: 7B
Bin->Dez: 123
Hex->Dez: 123
```

Mit Hilfe der Zahlenkonvertierung läßt sich nun auch schlüssig der Zusammenhang zwischen Halloween (31. Oktober) und Weihnachten (25. Dezember) belegen:

```
01 okt=31
02 dec=$(echo "obase=10;ibase=8;$okt" | bc)
03 echo "$okt.Okt. = $dec.Dez."
```

Die Ausgabe lautet wie erwartet: `31.Okt. = 25.Dez.`.

Zahlenumwandlung mit *printf*

Was aber, wenn kein *bc* zur Verfügung steht, weil er auf dem Kundenserver nicht installiert wurde? Nachinstallation auf einem fremden System geht ja nun mal nicht. Wenn eine *bash* verfügbar ist, hilft bei der Umwandlung von Dezimalwerten in Hex-Zahlen die Funktion `printf`. Diese Zeilen definieren die Umwandlung als Shell-Funktion. Der Aufruf `X=$(Hex 123) ; echo $X` liefert dann wunschgemäß den Wert *7B*:

```
01 Hex() # (Val)
02   {
03   VAL=$1
04   printf "%0X" $VAL
05   }
```

Bei der Umwandlung von Dezimalzahlen in Binärwerte ist es nicht ganz so einfach. Hier gilt es, wirklich zu programmieren. Es kommt das Horner-Schema für die Zahlenumwandlung zum Einsatz, das prinzipiell die Konvertierung beliebiger Zahlensysteme erlaubt. Das Horner-Schema dient vor allem zum Berechnen von Polynomen. Hier tritt die Zahlenbasis an die Stelle der Unbekannten, und die ermittelten Ziffern sind Koeffizienten des Polynoms. Klar wird das bei der Polynomschreibweise einer Zahl, etwa $3514 = 3 * 10^3 + 5 * 10^2 + 1 * 10^1 + 4$.

Um beispielsweise die Dezimalzahl *123* ins Binärsystem umzuwandeln, hilft folgendes Schema:

- Die Zahl wird durch 2 geteilt mit Rest.
- Der Divisionsrest wird notiert.
- Ist der ganzzahlige Quotient = 0, ist man fertig, andernfalls nimmt man ihn als neue Zahl und wiederholt den Durchlauf.

Das heißt also:

123 : 2 = 61	Rest: 1
61 : 2 = 30	Rest: 1
30 : 2 = 15	Rest: 0
15 : 2 = 7	Rest: 1
7 : 2 = 3	Rest: 1
3 : 2 = 1	Rest: 1
3 : 2 = 1	Rest: 1
1 : 2 = 0	Rest: 1

Um die endgültige Binärzahl zu erhalten, schreibt man die Divisionsreste beginnend von letzen Wert (unten) bis zum ersten Wert (oben) nieder. Die Berechnung liefert dann für das Beispiel 1111011.

Ein Shellskript realisiert das Aufschreiben von unten nach oben dadurch, dass der Programmierer den neu ermittelten Rest stets vor das bisherige Resultat schreibt. Außerdem ersetzt er die Division durch 2 mit einem Schiebebefehl und die Ermittlung des Rests durch das Abschneiden der letzten Stelle mittels *UND*-Verknüpfung. Beide Operationen sind meist schneller als die Division. Im Übrigen kommt der geschilderte Algorithmus eins zu eins zur Anwendung.

Das folgende Skript definiert wieder eine Shell-Funktion (Zeile 01). Zeilen 03 und 04 initialisieren Startwert und Resultat. Die Schleife (Zeilen 05 bis 08) läuft durch, bis der Quotient 0 geworden ist. Zeile 06 ermittelt den Divisionsrest durch eine *UND*-Verknüpfung mit 1 und setzt die letzte Stelle vor die Ergebnis-Variable. Zeile 07 dividiert den Wert `VAL` per Rechtsschieben durch 2. Das Echo-Kommando in Zeile 09 gibt nur den Wert zurück – ohne Zeilenvorschub.

```
01 Binary() # (VAL)
02   {
03   VAL=$1
04   RESULT=""
05   while [ $VAL -ne 0 ] ; do
06      RESULT=$(( $VAL & 1 ))$RESULT
07      VAL=$(( $VAL >> 1 ))
08   done
09   echo -n $RESULT
10   }
```

Ein Probelauf zeigt die gewünschten Ergebnisse:

```
$ X=$(Binary 123) ; echo $X
1111011
$ X=$(Binary 65) ; echo $X
1000001
$ X=$(Hex 123) ; echo $X
7B
$ X=$(Hex 65) ; echo $X
41
```

. . . und Action: Netzwerkarithmetik

Die Zahlenspielereien taugen für viele Praxisbereiche. So lässt sich auch die Rechnerei mit IP-Adressen oder Netzmasken per Shell erledigen. `bc` wird nur für eine Funktion benötigt. Alles andere kann die Bash von sich aus.

Die folgende Funktion wandelt eine IPv4-Adresse, die in der üblichen Dotted-Quad-Notation vorliegt, in eine Ganzzahl um. Zum Aufteilen des Strings *a.b.c.d* in seine vier Bestandteile dient ein Trick: Der Punkt wird in Zeile 03 zum Feldtrenner (normalerweise als Whitespace definiert), woraufhin mittels `set` Zeile 04 die IP-Adresse zerlegt, sodass man ihr die Variablen $1 bis $4 zuweisen kann. Diese sind dann nur noch mit den entsprechenden Zweierpotenzen zu multiplizieren und schließlich zu addieren (Zeilen 05 bis 08).

```
01 IP2Dec() # dotted quad
02   {
03   IFS='.'
04   set $1
05   echo "($1 * 16777216) \
06         + ($2 * 65536) \
07         + ($3 * 256) + $4" | bc
08   }
```

Ein Probelauf liefert:

```
$ X=$(IP2Dec '1.2.3.4') ; echo $X
16909060

$ X=$(IP2Dec '192.168.3.44') ; echo $X
3232236332
```

Die folgende, zweite Funktion macht genau das Gegenteil. Sie wandelt einen 32-Bit-Wert in die Dottet-Quad-Notation um. Dazu extrahiert sie sukzessive die niederwertigen acht Bit durch eine bitweise *UND*-Verknüpfung mit *0xFF* und speichert sie in *IP* (Zeilen 03 und 06). Dann dividiert sie den Zahlenwert durch 256 (Zeilen 04 und 07). Eine String-Konkatenation hängt die Zahlen und die Trennpunkte hintereinander (Zeile 06).

```
01 Dec2IP()
02   {
03   IP=$(($1 & 0xFF))
04   NUM=$(($1 / 256))
05   for N in 3 2 1; do
06     IP=$((NUM & 0xff))'.'$IP
07     NUM=$((NUM / 256))
08   done
09   echo $IP
10   }
```

Wenn man für den Probelauf die oben ermittelten 32-Bit-Zahlen eingibt, erhält man wieder die ursprünglichen IP-Adressen:

```
S=$(Dec2IP 16909060) ; echo $S
1.2.3.4

S=$(Dec2IP 3232236332) ; echo $S
192.168.3.44
```

Die Funktion *Dec2IP()* offenbart eine schöne Eigenschaft der Bash. Sie kann nämlich neben der üblichen Arithmetik auch bitweise Verknüpfungen bilden (wie C, Perl oder andere Programmiersprachen). Für alle, die mit Bitoperationen nicht so vertraut sind, bietet der Kasten „Bit-Operationen in der Bash„ einen kleinen Ausflug in die Welt der Bits.

Exkurs: Bit-Operationen in der Bash

Der Operator `&` führt eine bitweise *UND*-Verknüpfung auf zwei Integerzahlen durch. Jedes Bit im Ergebnis ist nur dann 1, wenn beide entsprechenden Bits in den beiden Eingabevariablen 1 sind:

```
    0 1 0 1 0 1 1 0   0x56
&   0 0 1 1 0 1 0 0   0x34
    ---------------
    0 0 0 1 0 1 0 0   0x14
```

Der Operator `|` führt eine bitweise *ODER*-Verknüpfung auf zwei Integerzahlen durch. Jedes Bit im Ergebnis ist dann 1, wenn eines der entsprechenden Bits in den Eingabevariablen 1 ist:

```
    0 1 0 1 0 1 1 0   0x56
|   0 0 1 1 0 1 0 0   0x34
    ---------------
    0 1 1 1 0 1 1 0   0x76
```

Der Operator `^` führt eine bitweise *Exklusiv-ODER*-Verknüpfung auf zwei Integerzahlen durch. Jedes Bit im Ergebnis ist genau dann 1, wenn eines, aber nicht beide der entsprechenden Bits in den beiden Eingangsoperanden 1 ist:

```
    0 1 0 1 0 1 1 0   0x56
 ^  0 0 1 1 0 1 0 0   0x34
    ---------------
    0 1 1 0 0 0 1 0   0x62
```

Der Operator führt ein bitweises Komplement auf einer einzelnen Integerzahl durch. (er ist also ein unärer Operator). Alle 0-Bits werden zu 1-Bits und umgekehrt:

```
 ~  0 1 0 1 0 1 1 0   0x56
    ---------------
    1 0 1 0 1 0 0 1   0xA9
```

Bei der Bash offenbart sich hier ein scheinbarer Bug:

```
$ echo $((~255))
-256
```

Ups! 255 stellt sich binär als 11111111 dar. Das Ergebnis sollte also *0000000* sein, und nicht *-256*. Wenn man aber bedenkt, dass der Zahlenbereich der Bash eben nicht 8-Bit-Zahlen, sondern 32- oder 64-Bit-Werte umfasst, wird der scheinbare Fehler klar.

```
  255 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
~ 255 = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
```

Letzteres ist eine negative Zahl, nämlich -256. Beim Negieren muss also gegebenenfalls der Zahlenbereich eingeschränkt werden:

```
$ echo $((~255 & 0xff))
0
```

Der Operator « verschiebt seinen ersten Operanden um die Anzahl von Bits nach links, die sein zweiter Operand angibt. Von rechts wird mit 0-Bits aufgefüllt. Analog verschiebt der Operator » den ersten Operanden nach rechts. Ist der erste Operand eine positive Zahl, werden 0-Bits von links aufgefüllt. Ist der erste Operand dagegen negativ, wird mit 1-Bits aufgefüllt:

```
    0 0 0 1 0 1 0 1 1 0  << 2   0x56
    -------------------
    0 1 0 1 0 1 1 0 0 0         0x158
```

Die folgenden Netzwerk-Rechenfunktionen stützen sich auf diese Bit-Operationen und auf die oben definierten Funktionen zur Konvertierung zwischen Dotted Quad und Integerzahl. Die öfters auftauchende Konstante `0xffffffff` stellt eine Binärzahl mit 32 1-Bits dar. Alle drei Funktionen haben einen String als Eingabeparameter, der eine IP-Adresse in *CIDR*-Notation enthält (`a.b.c.d/n`). Auch verzichten die Funktionen aus Gründen der Übersichtlichkeit auf die Überprüfung der Korrektheit des Eingabeparameters.

Die erste der folgenden drei Funktionen ermittelt die Netzmaske aus dem Eingabeparameter. Hier ist die Angabe der IP-Adresse nicht nötig, sondern nur die Angabe der Netzmasken-Bits. Damit aber alle drei Funktionen mit einem einheitlichen Parameterformat versorgt werden, übergibt das Listing auch hier die IP-Adresse.

Die Funktion fischt sich in Zeile 03 mittels `cut` den benötigten Teil heraus. Da die Netzmaske für den Netzwerkanteil der IP-Adresse lauter 1-Bits und für den Host-Anteil lauter 0-Bits enthält, schiebt Zeile 04 einfach eine 32-Bit-Zahl aus lauter 1-Bits um die entsprechende Anzahl von Host-Bits nach links. Dabei werden automatisch 0-Bits von rechts nachgeschoben. Die oben definierte Funktion `Dec2IP()` erzeugt Zeile 05 dann aus der Maske das Dotted Quad:

```
01: NETMASK() # IP-Addr/Netmask
02:   {
03:   MASK=$(echo $1 | cut -d'/' -f2)
04:   MASK=$((0xffffffff << (32 - $MASK)))
05:   Dec2IP $MASK
06:   }
```

Auf ähnliche Art ermittelt man die Netzwerkadresse. Hier ist auch die IP-Adresse nötig, die man aus dem Parameter per `cut` abtrennt und mittels `IP2Dec` vom Dotted Quad zu einer 32-Bit-Integerzahl konvertiert (Zeilen 03 und 04). Danach berechnet die Funktion die Netzmaske wie schon in der vorhergehenden Funktion (Zeilen 05 und 06). Zeile 07 *UND*-verknüpft die IP-Adresse mit der Netzmaske, was in der Folge alle Bits des Host-Anteils der IP-Adresse auf 0 setzt. Das Ergebnis erscheint ebenfalls dank Zeile 07 auch wieder als Dotted Quad:

```
01 NETWORK() # IP-Addr/Netmask
02   {
03   ADDR=$(echo $1 | cut -d'/' -f1)
04   ADDR=$(IP2Dec $ADDR)
05   MASK=$(echo $1 | cut -d'/' -f2)
06   MASK=$((0xffffffff << (32 - $MASK)))
07   Dec2IP $(($ADDR & $MASK))
08   }
```

Die folgende, dritte Funktion schließlich berechnet die Broadcast-Adresse. Auch sie extrahiert, wie oben beschrieben, die Adresse und die Netzmaske aus den Parametern (Zeilen 03 bis 06). Um die Broadcast-Adresse zu erhalten, muss man dann die IP-Adresse mit der invertierten Netzmaske *ODER*-verknüpfen. Zeile 06 invertiert die Netzmaske korrekt, wobei hier die weiter oben geschilderte Komplementdarstellung zu berücksichtigen ist. Zeile 08 erledigt die *ODER*-Verknüpfung und es gibt das Ergebnis als Dotted Quad zurück:

```
01 BROADCAST() # IP-Addr/Netmask
02   {
03   ADDR=$(echo $1 | cut -d'/' -f1)
04   ADDR=$(IP2Dec $ADDR)
05   MASK=$(echo $1 | cut -d'/' -f2)
06   MASK=$((0xffffffff << (32 - $MASK)))
07   MASK=$((~$MASK & 0xffffffff))
08   Dec2IP $(($ADDR | $MASK))
09   }
```

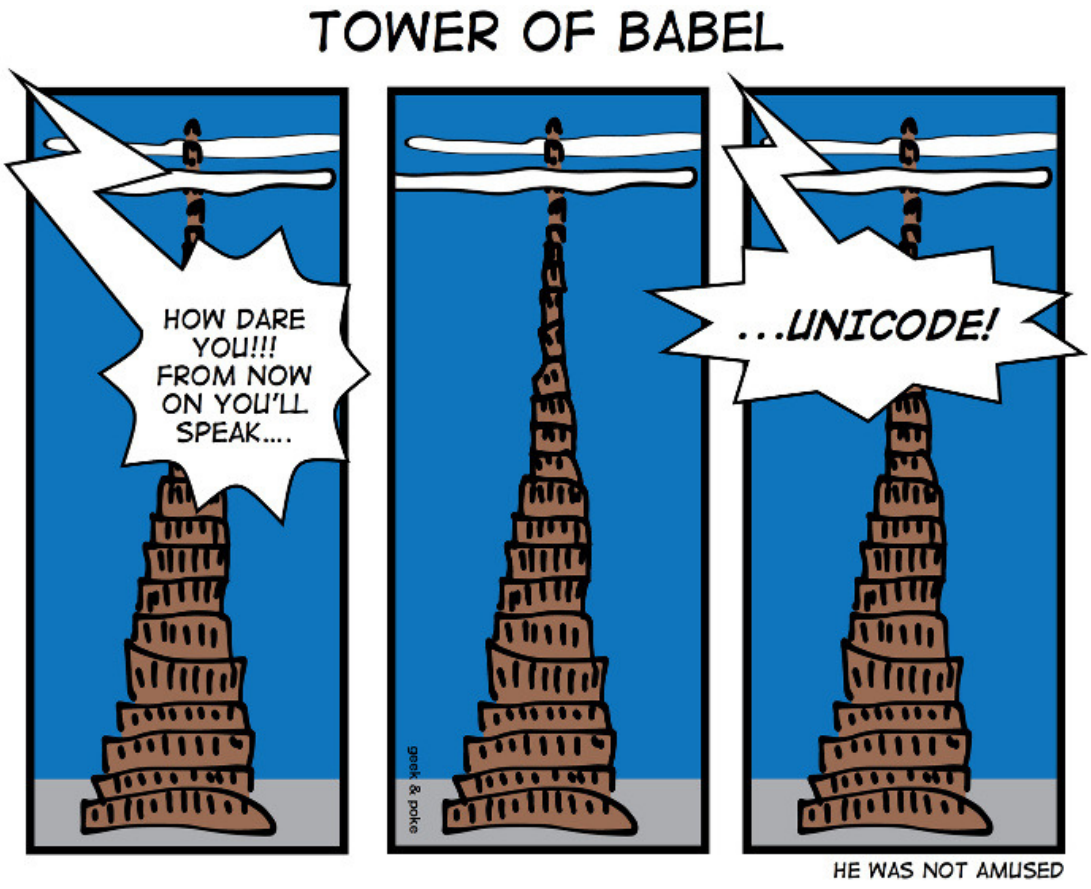
Das folgende Protokoll demonstriert die drei Funktionen:

```
S=$(NETMASK '192.168.3.44/24')
echo $S
255.255.255.0

S=$(NETWORK '192.168.3.44/24')
echo $S
192.168.3.0

S=$(BROADCAST '192.168.3.44/24')
echo $S
192.168.3.255
```

Nun steht dem Generieren von `ip`-, `ifconfig`- oder `route`-Kommandos sowie dem Erzeugen einer `interfaces`-Datei per Shellskript nichts mehr im Wege.

Über Jürgen

Jürgen Plate ist Professor für Elektro- und Informationstechnik an der *Hochschule für angewandte Wissenschaften München*. Er beschäftigt sich seit 1980 mit Datenfernübertragung und war, bevor der Internetanschluss für Privatpersonen möglich wurde, in der Mailboxszene aktiv. Unter anderem hat er eine der ersten öffentlichen Mailboxen – TEDAS der *mc*-Redaktion – programmiert und 1984 in Betrieb genommen.



Hilfreiches für alle Beteiligten
Autorenrichtlinien

Selbst etwas für die UpTimes schreiben? Aber ja! Als Thema ist willkommen, was ein GUUG-Mitglied interessiert und im Themenbereich der GUUG liegt. Was sonst noch zu beachten ist, steht in diesen Autorenrichtlinien.

Der Schriftsteller ragt zu den Sternen empor,
Mit ausgefranstem T-Shirt.
Er raunt seiner Zeit ihre Wonnen ins Ohr,
Mit ausgefranstem T-Shirt.

Frei nach Frank Wedekind, Die Schriftstellerhymne

Wir sind an Beiträgen interessiert. Wir, das ist diejenige Gruppe innerhalb der GUUG, die dafür sorgt, dass die UpTimes entsteht. Dieser Prozess steht jedem GUUG-Mitglied offen. Der Ort dafür ist die Mailingliste <redaktion@uptimes.de>.

Welche Themen und Beitragsarten kann ich einsenden?

Die UpTimes richtet sich als Vereinszeitschrift der GUUG an Leser, die sich meistens beruflich mit Computernetzwerken, IT-Sicherheit, Unix-Systemadministration und artverwandten Themen auseinandersetzen. Technologische Diskussionen, Methodenbeschreibungen und Einführungen in neue Themen sind für dieses Zielpublikum interessant, Basiswissen im Stil von *Einführung in die Bourne Shell* hingegen eher nicht. Wer sich nicht sicher ist, ob sein Thema für die UpTimes von Interesse ist, kann uns gern eine E-Mail an <redaktion@uptimes.de> schicken.

Neben Fachbeiträgen sind Berichte aus dem Vereinsleben, Buchrezensionen, Konferenzberichte, humoristische Formen und natürlich Leserbriefe interessant. Wer nicht gleich mehrseitige Artikel schreiben möchte, beginnt also mit einem kleineren Beitrag.

Fachbeiträge sind sachbezogen, verwenden fachsprachliches Vokabular und anspruchsvolle Erläuterungen, besitzen technische Tiefe und ggf. auch Exkurse. Berichte aus dem Vereinsleben greifen aktuelle Themen auf oder legen Gedankengänge rund um die GUUG und ihre Community dar. Konferenzberichte zeigen, welche Veranstaltungen jemand besucht hat, was er/sie dort erfahren hat und ob die Veranstaltung nach Meinung des Autoren beachtenswert oder verzichtbar war. Unterhaltsame Formen können ein Essay oder eine Glosse sein, aber auch Mischformen mit Fachartikeln (Beispiel: der „Winter-Krimi“ in Ausgabe 2013-3). Auch unterhaltsame Formen besitzen jedoch inhaltlichen Anspruch. Denn die UpTimes ist und bleibt die Mitgliederzeitschrift eines Fachvereines.

In der UpTimes legen wir daher auch Wert auf professionelle publizistische Gepflogenheiten und einheitliche Schreibweisen. Dafür sorgt zum Beispiel ein einheitliches Layout der Artikel, oder etwa die grundsätzliche Vermeidung von Worten in Großbuchstaben (entspricht typografisches Schreien) oder von Worten in Anführungsstrichen zum Zeichen der Uneigentlichkeit (entspricht Distanzierung von den eigenen Worten). Wichtig sind außerdem beispielsweise Quellenangaben bei Zitaten, Kenntlichmachung fremder Gedanken, Nachvollziehbarkeit der Argumentation sowie Informationen zum Autor nach dem Artikel.

In welchem Format soll ich meinen Artikel einsenden?

ASCII: Am liebsten blanke UTF8-Texte. Gern mit beschreibenden Anmerkungen oder Hinweisen (zum Beispiel mit Prozentzeichen zum Kenntlichmachen der Meta-Ebene).

LaTeX: Wir setzen die UpTimes mit LaTeX. Weil wir – wie es sich beim Publizieren gehört – mehrspaltig setzen und ein homogenes Erscheinungsbild anstreben, verwenden wir für die UpTimes bestimmte Formatierungen. Es ist nicht erwünscht, eigene Layoutanweisungen einzusenden. Wir behalten uns vor, Texte für die Veröffentlichung in der UpTimes umzuformatieren. Eine Vorlage mit den von uns verwendeten Auszeichnungen für Tabellen, Kästen und Abbildungen gibt es unter http://www.guug.de/uptimes/artikelvorlage.tex.

Listings: Der mehrspaltige Druck erlaubt maximal 45 Zeichen Breite für Code-Beispiele, inklusive 1 Leerzeichen und einem Zeichen für den Zeilenumbruch innerhalb einer Code-Zeile (Backslash). Breitere Listings formatieren wir um, verkleinern die Schriftgröße oder setzen sie als separate Abbildung.

Bilder: Wir verarbeiten gängige Bildformate, soweit ImageMagick sie verdaut und sie hochauflösend sind. Am besten eignen sich PNG- oder PDF-Bilddateien. Plant bei längeren Artikeln mit 1 Abbildung pro 3000 Zeichen. Das müssen nicht Bilder sein, sondern auch Tabellen, Listings oder ein Exkurskasten sind möglich. Verseht Eure Bilder nicht mit Rahmen oder Verzierungen, weil die Redaktion diese im UpTimes-Stil selbst vornimmt.

Wie lang kann mein Artikel sein?

Ein einseitiger Artikel hat mit zwei Zwischentiteln um die 2.700 Anschläge. Mit etwa 15.000 Anschlägen – inklusive 3 Abbildungen – landet man auf rund vier Seiten. Wir nehmen gern auch achtseitige Artikel, achten dabei aber darauf, dass der Zusammenhang erhalten bleibt und dass es genug Bilder gibt, damit keine Textwüsten enstehen.

Wer Interesse hat, für die UpTimes zu schreiben, macht sich am besten um die Zeichenzahl nicht so viele Gedanken – auch für kurze oder lange Formate finden wir einen Platz. Die Redaktion ist bei der konkreten Ideenentwicklung gern behilflich. Für eine Artikelidee an <redaktion@uptimes.de> reicht es, wenn Ihr ein bestimmtes Thema behandeln wollt.

Wohin mit meinem Manuskript?

Am einfachsten per E-Mail an <redaktion@uptimes.de> schicken. Das ist jederzeit möglich, spätestens jedoch vier Wochen vor dem Erscheinen der nächsten UpTimes. Zum Manuskript ist ein kleiner Infotext zum Autor wichtig, ein Bild wünschenswert.

Nützlich ist, wenn der Text vor Einsendung durch eine Rechtschreibkorrektur gelaufen ist. `aspell`, `ispell` oder `flyspell` für Textdateien sowie die von LibreOffice bieten sich an. Wenn Ihr Euren Text an die Redaktion schickt, solltet Ihr also weitestmöglich bereits auf die Rechtschreibung geachtet haben: Nach der Einschickung ist Rechtschreibung und Typo-Korrektur Aufgabe der Redaktion. Die Texte in der UpTimes folgen der neuen deutschen Rechtschreibung.

Wie verlaufen Redaktion und Satz?

Wir behalten uns vor, Texte für die Veröffentlichung in der UpTimes zu kürzen und zu redigieren. Das bedeutet, dafür zu sorgen, dass der Artikel nicht ausufert, versehentliche Leeraussagen wegfallen, Syntax und Satzanschlüsse geglättet werden, dass Passiva und Substantivierungen verringert und Unklarheiten beseitigt werden (die zum Beispiel Fragen offen lassen oder aus Passivkonstruktionen resultieren, ohne dass der Schreibende das merkt). Manchmal ist dieser Prozess mit Nachfragen an den Autoren verbunden.

Die endgültige Textversion geht jedem Autoren am Ende zur Kontrolle zu. Dabei geht es um die inhaltliche Kontrolle, ob sich durch den Redaktionsprozess Missverständnisse oder Falschaussagen entstanden sind. Danach setzt die Redaktion die Artikel. Wenn der Satz weitgehend gediehen ist – also ein *Release Candidate* als PDF vorliegt – erhalten die Autoren als erste diesen RC. Danach wird die UpTimes dann veröffentlicht.

Gibt es Rechtliches zu beachten?

Die Inhalte der UpTimes stehen ab Veröffentlichung unter der CC-BY-SA-Lizenz, damit jeder Leser die Artikel und Bilder bei Nennung der Quelle weiterverbreiten und auch weiterverarbeiten darf. Bei allen eingereichten Manuskripten gehen wir davon aus, dass der Autor sie selbst geschrieben hat und der UpTimes ein nichtexklusives, aber zeitlich und räumlich unbegrenztes Nutzungs- und Bearbeitungsrecht unter der CC-BY-SA einräumt.

Bei Fotos oder Abbildungen Dritter ist es rechtlich unabdingbar, dass der Autor sich bei dem Urheber die Erlaubnis zu dieser Nutzung einholt, und fragt, wie die Quelle genannt zu werden wünscht. Die Frage nach der CC-BY-SA ist hierbei besonders wichtig.

An Exklusivrechten, wie sie bei kommerziellen Fachzeitschriften üblich sind, hat die UpTimes kein Interesse. Es ist den Autoren freigestellt, ihre Artikel noch anderweitig nach Belieben zu veröffentlichen.

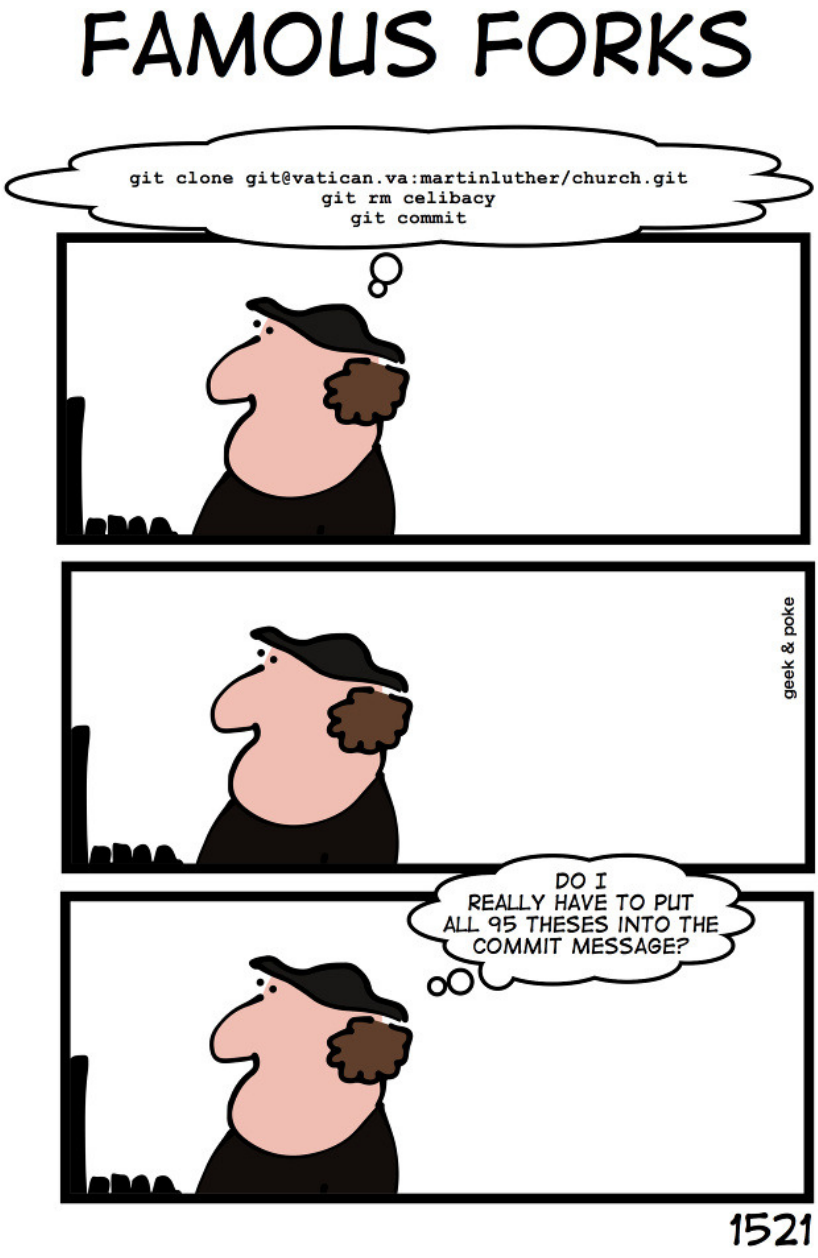
Bekomme ich ein Autorenhonorar?

Für Fach- und literarische Beiträge zahlt die GUUG dem Autor nach Aufforderung durch die Redaktion und Rechnungstellung durch den Autor pro Seite 50 € zuzüglich eventuell anfallender USt. Beiträge für die Rubrik „Vereinsleben“,

Buchrezensionen und Artikel bezahlter Redakteure sind davon ausgenommen. Gleiches gilt für Paper, wenn die UpTimes die Proceedings der Konferenz enthält.

Nächste redaktionelle Ausgabe:
UpTimes 2017-1, Sommer-Ausgabe

- Redaktionsschluss: Sonntag, 14. Mai 2017.
- Erscheinung: Sonntag, 18. Juni 2017.
- Gesuchte Inhalte: Fachbeiträge über Unix und verwandte Themen, Veranstaltungsberichte, Rezensionen, Beiträge aus dem Vereinsleben.
- Manuskript-Template: http://www.guug.de/uptimes/artikel-vorlage.tex
- Fragen, Artikelideen und Manuskripte an: <kehrer@guug.de> oder an die Redaktionsmailingliste <redaktion@uptimes.de>



Über die GUUG
German Unix User Group e.V.

Vereinigung deutscher Unix-Benutzer

Die Vereinigung Deutscher Unix-Benutzer hat gegenwärtig rund 700 Mitglieder, davon etwa 90 Firmen und Institutionen.

Im Mittelpunkt der Aktivitäten der GUUG stehen Konferenzen. Ein großes viertägiges Event der GUUG hat eine besondere Tradition und fachliche Bedeutung: In der ersten Jahreshälfte treffen sich diejenigen, die ihren beruflichen Schwerpunkt im Bereich der IT-Sicherheit, der System- oder Netzwerkadministration haben, beim *GUUG-Frühjahrsfachgespräch* (FFG).

Seit Oktober 2002 erscheint mit der *UpTimes* – die Sie gerade lesen – eine Vereinszeitung. Seit 2012 erscheint die UpTimes einerseits zu jedem FFG in Form einer gedruckten Proceedings-Ausgabe (ISBN), und andererseits im Rest des Jahres als digitale Redaktionsausgabe (ISSN). Daneben erhalten GUUG-Mitglieder zur Zeit die Zeitschrift *LANline* aus dem Konradin-Verlag kostenlos im Rahmen ihrer Mitgliedschaft.

Schließlich gibt es noch eine Reihe regionaler Treffen (http://www.guug.de/lokal): im Rhein-Ruhr- und im Rhein-Main-Gebiet sowie in Berlin, Hamburg, Karlsruhe und München.

Warum GUUG-Mitglied werden?

Die GUUG setzt sich für eine lebendige und professionelle Weiterentwicklung im Open Source-Bereich und für alle Belange der System-, Netzwerkadministration und IT-Sicherheit ein. Wir freuen uns besonders über diejenigen, die bereit sind, sich aktiv in der GUUG zu engagieren. Da die Mitgliedschaft mit jährlichen Kosten

Fördermitglied	350 €
persönliches Mitglied	70 €
in der Ausbildung	30 €

verbunden ist, stellt sich die Frage, welche Vorteile damit verbunden sind?

Neben der Unterstützung der erwähnten Ziele der GUUG profitieren Mitglieder auch finanziell davon, insbesondere durch die ermäßigten Gebühren bei den Konferenzen der GUUG und denen anderer europäischer UUGs. Mitglieder bekommen außerdem *c't* und *iX* zum reduzierten Abopreis.

Wie GUUG-Mitglied werden?

Füllen Sie einfach das Anmeldeformular unter https://www.guug.de/verein/mitglied/ aus und schicken Sie es per Fax oder Post an die unten auf dem Formular angegebene Adresse.

Impressum

Uptimes – Mitgliederzeitschrift der
German Unix User Group (GUUG) e.V.
Herausgeber: GUUG e.V.
Grube-Nassau-Straße 3
D-56462 Höhn
E-Mail: <redaktion@uptimes.de>
Internet: http://www.guug.de/uptimes/

Autoren dieser Ausgabe: Anika Kehrer, Nils Magnus, Hanno Wagner, Lenz Grimmer, Mathias Weidner, Jürgen Plate
V.i.S.d.P: Anika Kehrer
Oslostr. 9
D-81829 München
Chefredaktion: Anika Kehrer
Redaktion: Mathias Weidner
LaTeX-Layout (PDF): Robin Därmann
XHTML-Layout (ePub): Mathias Weidner
Titelgestaltung: Hella Breitkopf
Bildnachweis: Titelbild *Raureif-Nadeln* von Hella Breitkopf. Comicreihe *geek & poke* von Oliver Widder, CC-BY-SA, mit zusätzlicher freundlicher Genehmigung von Oliver Widder. Andere Quellennachweise am Bild, sofern nicht unter CC BY SA.
Verlag: Lehmanns Media GmbH, Hardenbergstraße 5, 10623 Berlin
ISSN: 2195-0016

Für Anzeigen in der UpTimes wenden Sie sich bitte an <werbung@guug.de>.