Some Bashing

Linux-Infotag Augsburg

28. März 2015

Andreas Steil
Linux Consultant & Trainer
B1 Systems GmbH
steil@b1-systems.de

Some Bashing – Agenda

- Allgemeine Grundlagen: Unix/GNU/Linux, Shells
- Bash: Features/Möglichkeiten
- Textwerkzeuge: `cat`, `cut`, `grep`, `sed`, `wc`, ...
- Internetwerkzeuge: `wget`, `lynx`, `mutt`, `tcpdump`, ...
- Beispiele:
  - Einzeiler aus der Praxis
  - Doppelte Dateien finden
  - PDF-Dokumente zusammenführen
  - Dateien konvertieren
  - Surfen im Netz
  - Zufällige Spielereien
  - Spam or Not Spam?
  - Bash Forkbomb
  - Suche Amelie

Etwas (Unix-)Philosophie

Douglas McIlroy (Erfinder der Unix-Pipes):

- Schreibe Computerprogramme so, dass sie nur eine Aufgabe erledigen und diese gut machen.
- Schreibe Programme so, dass sie zusammenarbeiten.
- Schreibe Programme so, dass sie Textströme verarbeiten, denn das ist eine universelle Schnittstelle.

Unix-Philosophie (verkürzt):

"Mache nur eine Sache und mache sie gut."

Mike Gancarz: The UNIX Philosophy (1994):

- **Klein ist schön.**
- **Gestalte jedes Programm so, dass es eine Aufgabe gut erledigt.**
- Erzeuge so bald wie möglich einen funktionierenden Prototyp.
- Bevorzuge Portierbarkeit vor Effizienz.
- **Speichere Daten in einfachen Textdateien.**
- Verwende die Hebelwirkung der Software zu deinem Vorteil.
- Verwende Shell-Skripte, um die Hebelwirkung und die Portierbarkeit zu verbessern.
- **Vermeide Benutzeroberflächen, die den Benutzer fesseln.**
- **Mache jedes Programm zu einem Filter.**

. . . und noch ein Zitat:

„Unix ist einfach. Es erfordert lediglich ein Genie, um seine Einfachheit zu verstehen.“ (Dennis Ritchie)

Was ist eine Shell?

- Shell = Schnittstelle zwischen Benutzer und Betriebssystem
- Kommandointerpreter, meist unter `/bin/`
- viele mögliche Shells, u. a.:

`/bin/sh`	ursprüngliche Bourne-Shell
`/bin/ash`	Almquist-Shell (Standard bei FreeBSD/NetBSD)
`/bin/bash`	„Bo(u)rne again shell“ (Standard bei GNU/Linux)
`/bin/csh`	C-Shell
`/bin/ksh`	Korn-Shell
	...

- Standard-Shell der meisten GNU/Linux-Systeme: Bash

Shell als Kommandointerpreter

- Shell ist ein Kommandointerpreter
- jede Eingabe wird durch die Shell „interpretiert“
- Sonderzeichen haben bestimmte Bedeutung
- allgemeiner Aufbau eines Shell-Kommandos:

```
<Befehl> [Option(en)] <Argument(e)>
```

- erstes „Wort“ der Eingabe ist der Befehl
- Leerzeichen oder Tabulator trennt Argumente in der Shell
- besonderer Typ von Argumenten:
  Optionen, eingeleitet durch - oder --
- Befehle/Pfade lassen sich mit der Tabulator-Taste ergänzen

Bash-Features – ein Überblick

- History
- Eingabekomplettierung
- Ein- & Ausgabeumleitung
- Piping
- Aliasing
- Befehlsverkettung
- Kommandosubstitution
- Parameter (z. B. `$@`)
- Textwerkzeuge
- Skripting, Funktionen
- Aufruf anderer Interpreter (Perl, Python, . . .)
- umfassendes Hilfesystem (Man-Pages, u.v.a.)
- . . .

Metazeichen (Sonderzeichen) der Bash

Leerzeichen	Trennzeichen zwischen Argumenten
Tabulator	Trennzeichen zwischen Argumenten
/	Trennzeichen bei Pfadangaben
$	Variable, z. B. `$HOME`
*	beliebige Zeichen, z. B. `datei*`
?	genau *ein* beliebiges Zeichen, z. B. `datei?`
\	Maskierung, z. B. `neue\ datei`
", '	Maskierung, z. B. `"neue datei"`
<	Umleitung Standardeingabe
>	Umleitung Standardausgabe
|	„Pipe“-Zeichen zur Weiterverarbeitung
;	Kommando abschließen
!	Zugriff auf History; Verneinung

. . . z. B. Metazeichen bei Verzeichnisangaben:

```
# cd /
```

⇒ wechselt ins Root-Directory (/ alleinstehend)

```
# bash ./skripte/mach.sh
```

⇒ ausgehend vom aktuellen Verzeichnis (.)

```
# cd ..
```

⇒ wechselt ins nächsthöhere Verzeichnis

```
# cd -
```

⇒ wechselt ins vorhergehende Verzeichnis

```
# cd ~
```

⇒ wechselt ins Home-Directory (nur `cd` ohne Tilde geht auch)

Kommandoverknüpfung

| „Pipe“-Zeichen zur Weiterverarbeitung
; Trennung – Kommandos nacheinander ausführen
&& wenn voriger Befehl erfolgreich (Exit Code = 0 / `TRUE`)
|| wenn voriger Befehl nicht erfolgreich (Exit Code = 1 / `FALSE`)

Einfache Kommandoverknüpfung mit ||:

```
# ls xyuzgzg || echo gibds ned
ls: Zugriff auf xyuzgzg nicht möglich: \
Datei oder Verzeichnis nicht gefunden
gibds ned
```

Einfache Kommandoverknüpfung mit || – schöner:

```
# ls xyuzgzg 2> /dev/null || echo $_ gibd\'s ned
xyuzgzg gibd's ned
```

Kommandosubstitution

Einfache Kommandosubstitution mit ‘...‘:

```
# echo ‘date +%H:%M‘
10:30
```

Einfache Kommandosubstitution mit $(...) und Aliasing:

```
# alias uhr="echo $(date +%H:%M)"; uhr
10:33
```

Kommandosubstitution bei Dateinamen:

```
# cp foo.txt "foo--$(date +"%Y-%m-%d").txt"; ls foo--*
```

Etwas komplexere Kommandosubstitution zur Datensicherung:

```
# find /daten -type f -newermt 2015-03-25 \
  -exec cp -a {} /backup/ \;
```

⇒ weitere Automatisierung durch `crontab`, Startskripte ...

Rechnen mit der Bash

Rechnen mit expr:

```
# a=2; b=3; c=$(expr $a + $b); echo $c
5
```

Kommazahlen mit bc:

```
# a=1.23;b=-2.34; echo "$a*$b" | bc
-2.87
```

Kommazahlen mit bc und 5 Nachkommastellen:

```
# a=5; b=3; c=‘echo "scale=5; $a / $b" | bc‘; echo $c
1.66666
```

Anwendung als Zähler (z. B. in Skripten):

```
typeset -i i=0; ...; i=$i+1 # Variante 1
i=‘expr $i + 1‘             # Variante 2
i=$(($i + 1))               # Variante 3
let "i += 1"                # Variante 4
```

Textwerkzeuge – Überblick

- Einfache Textwerkzeuge:
  - `cat`
  - `cut`
  - `diff`
  - `head`
  - `tail`
  - `tr`
  - `uniq`
  - `wc`
  - `...`
- Komplexe Textwerkzeuge:
  - `grep`
  - `sed`
  - `awk`

Textwerkzeuge – grep

- grep = Programm zur Suche und Filterung definierter Zeichenketten in Dateien dient
- ursprünglich von Ken Thompson entwickelt
- steht für „global/regular expression/print“

grep – allgemeine Syntax:

```
$ grep [optionen] <Muster> [Datei(-liste)]
```

grep – Beispiele:

```
# grep kernel /var/log/messages > kernelmessages
# grep ^[^#] /etc/sysconfig/kernel
# seq 1 99 | grep -E "(1|2)" | wc -l
```

Textwerkzeuge – sed

- „Stream Editor“ (kein interaktiver Editor wie `vi`)
- Datei oder Standardeingabe wird zeilenweise abgearbeitet
- Basismodus: Nur erste Fundstelle jeder Zeile wird gewertet

sed-Aufruf

```
$ sed '<befehl>' <datei>
```

- Editierbefehle immer ein Buchstabe
- bearbeitete Zeilen werden auf Standardausgabe ausgegeben

sed – Editierbefehle

Einige Editierbefehle von sed

Befehl	Funktion
a	Einfügen nach der aktuellen Zeile
i	Einfügen vor der aktuellen Zeile
d	Löschen
p	Ausgeben
c	Zeilen ersetzen
s	Suchen und Ersetzen
y	Zeichen durch andere Zeichen ersetzen

sed – Suchen & Ersetzen

- eine der wichtigsten Funktionen von sed: Suchen & Ersetzen
- Suchbegriff meist regulärer Ausdruck
- nur erstes Vorkommen pro Zeile wird ersetzt, es sei denn, Sie verwenden Parameter g (*global*)

Erstes Vorkommen pro Zeile wird ersetzt

```
$ sed 's/Latex/LaTeX/' datei
```

Alle Vorkommen werden ersetzt

```
$ sed 's/Latex/LaTeX/g' datei
```

Anwendung von sed – Beispiele

Beispiel: Zeile 12 bis Ende der Datei löschen

```
$ sed '12,$d' datei
```

Beispiel: Zeile 5 durch „lalala“ ersetzen

```
$ sed '5c lalala' datei
```

Bestimmte Buchstaben ersetzen

```
$ echo "P1 Sysdems" | sed -e 'y/Pd/Bt/'
```

(Konfigurations-)Dateien suchen und Kommentare entfernen

```
# find /<pfad>/ -type f -iname "*.conf" -exec sed -i '/^#/d' '{}' \;
```

Textwerkzeuge – awk

- Programmiersprache (Skriptsprache) zur Bearbeitung und Auswertung strukturierter Textdaten
- Name stammt von Autoren *Aho*, *Weinberger*, *Kernighan*
- ursprünglicher Zweck: Reports aus textbasierten Datenbankdateien generieren (CSV-Dateien u.ä.)
- arbeitet mit dem Datentyp Zeichenkette (*String*)
- Syntax ähnelt der Programmiersprache C
- eine Version von `awk` auf fast jedem Unix-System
- grundlegende Bestandteile der Sprache:
  - assoziative Arrays (mit Zeichenketten indizierte Arrays, „Hashes“)
  - reguläre Ausdrücke
- Leistungsfähigkeit, Kompaktheit, aber auch Beschränkungen der awk- und sed-Skripte regten Larry Wall zur Entwicklung der Sprache Perl an.

awk – Beispiele

Aufbau von Befehlen

```
awk '<Bedingung> {<Aktion>}'
```

Standard-„Aktion“: print

```
$ awk '/tux/ {print}' /etc/passwd
tux:x:1000:100:Tux Pinguin:/home/tux:/bin/bash
```

3. Feld von jeder Zeile, die mit A beginnt und deren zweites Feld eine Zahl größer 42 ist:

```
$ awk $1 ~ /^A/ && $2 > 42 { print $3 }
```

Weitere Beispiele zur Anwendung von Textwerkzeugen

cat mit kombinierter Ein-/Ausgabeumleitung (»Here Documents«):

```
$ cat << eof > textdatei
abc
eof
```

Home-Directory eines Benutzers ermitteln mit cut:

```
# cat /etc/passwd | grep b1 | cut -d: -f 6
/home/b1
```

. . . etwas komfortabler:

```
# alias wodahoam="echo -n Welcher Benutzer \?\ ; \
read var; cat /etc/passwd | grep $var | cut -d: -f 6"
# wodahoam
Welcher Benutzer ? b1
/home/b1
```

Internet-Werkzeuge

Browser	HTTP/S (`lynx`, `w3m`, ...)
Dateitransfer	FTP (`ftp`, `scp`), HTTP/S (`wget`, `curl`)
Mail	POP/IMAP, SMTP (`mail`, `mutt`, `ssmtp`, ...)
Jabber	XMPP (`mcabber`, `sendxmpp`, ...)
Abfragen	DNS (`dig`, `whois`, ...)
Analyse	alle Protokolle (`arp`, `tcpdump`, `tracert`, `netstat`, `nmap`, ...)

Beispiel: Dateien im Netz übertragen

- `wget` Kommandozeilenprogramm des GNU-Projekts zum Herunterladen von Dateien; unterstützte Protokolle: HTTP/S, FTP

Datei-Download mit `wget` (HTTP/S):

```
# wget https://raw.github.com/openstack/nova/master/etc\
      /nova/logging_sample.conf \
      -O /etc/nova/logging.conf
```

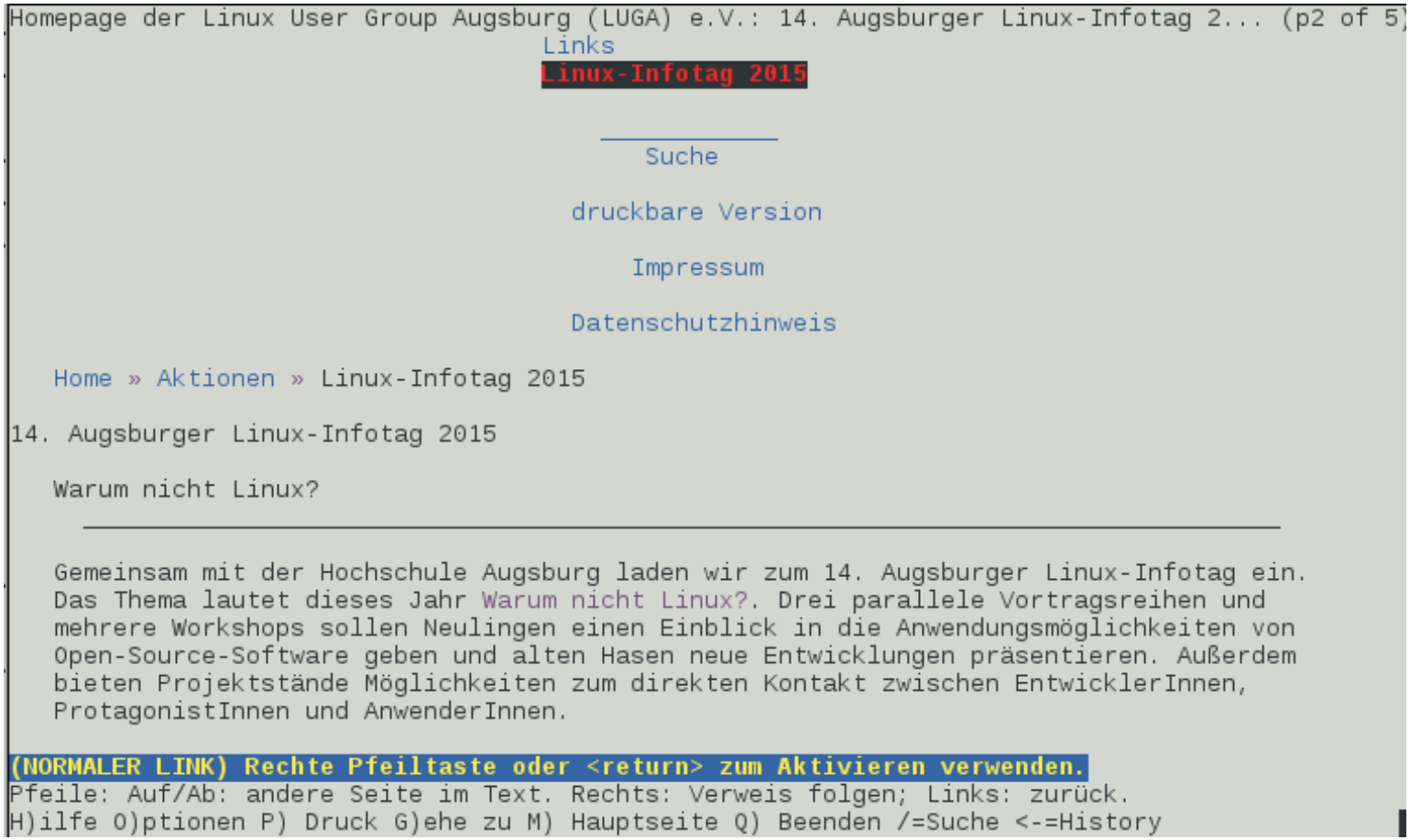
- *Secure CoPy* (SCP) = Protokoll / Programm zur verschlüsselten Datenübertragung, basiert auf *Secure SHell* (SSH)

Datei-Transfer mit `scp` (SSH):

```
# scp <benutzer>@<host>:<quelldatei> <ziel>
# scp <quelldatei> <benutzer>@<host>:<ziel>
```

Beispiel: Surfen im Netz mit `lynx` (HTTP/S)

```
# lynx http://www.luga.de/Aktionen/LIT-2015/
```



Beispiel: Netzwerkverkehr aufzeichnen mit `tcpdump`

Ein einfacher `ping` ...

```
#  ping -c 1 heise.de
```

...und seine Folgen (Aufzeichnung mit `tcpdump`):

```
# tcpdump -i wlp3s0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlp3s0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:30:20.173459 IP 192.168.178.1.53 > 192.168.178.27.36086: 64621 1/0/0 A 213.5.173.62 (48)
07:30:20.173604 IP 192.168.178.27.43318 > 192.168.178.1.53: 40891+ AAAA? www.alahli.com. (32)
07:30:20.378482 IP 192.168.178.1.53 > 192.168.178.27.43318: 40891 0/1/0 (97)
07:30:23.127836 IP 192.168.178.27 > 193.99.144.80: ICMP echo request, id 32746, seq 1, length 64
07:30:23.157748 IP 193.99.144.80 > 192.168.178.27: ICMP echo reply, id 32746, seq 1, length 64
07:30:24.705061 IP 192.168.178.27.41364 > 185.48.119.46.5222: Flags[P.],seq 1116239838:1116239936,\
                ack 158396364, win 1414, options [nop,nop,TS val 106152003 ecr1051851078], length 98
[...]
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

Beispiele: Einzeiler aus der Praxis

Schreibgeschwindigkeit testen

```
# time $(dd if=/dev/zero of=testdatei bs=1000 count=100000 && sync)
```

Definition des Parameters „host_ip“ bei OpenStack finden

```
# find /usr/lib64/python2.6/site-packages/nova/ \
      -name "*py" | xargs grep host_ip
```

Ersetzen einer IP-Adresse in allen Dateien unter /etc

```
# for i in $(grep -rlIs "10.0.0.1" /etc); do \
  sed 's/10.0.0.1/192.168.42.1/' $i > sed.tmp; mv sed.tmp $i; done
```

Docker-Container betreten

```
# nsenter --target $(docker inspect -f '{{ .State.Pid }}' \
  $CONTAINER_ID) -m -u -i -n -p mount | head -1 | awk '{ print $1 }'
```

Beispiel: Digitaluhr mit Sekundenanzeige

Digitaluhr mit Sekundenanzeige (`while`-Schleife):

```
# while true; do echo ‘date +%T‘; sleep 1; clear; done
```

Digitaluhr mit Sekundenanzeige (als Skript):

```
#!/bin/bash
while true
do
   echo ‘date +%T‘
   sleep 1
   clear
done
```

Beispiel: Doppelte Dateien finden

Einmaligkeitstest mit `md5sum`:

```
# find ./ -exec md5sum {} 2>/dev/null \; | sort | uniq -w 32 -D
b026324c6904b2a9cb4b88d6d61c81d1  ./test1
b026324c6904b2a9cb4b88d6d61c81d1  ./test-uv/test1
d41d8cd98f00b204e9800998ecf8427e  ./foo.txt
d41d8cd98f00b204e9800998ecf8427e  ./test6
```

`-exec md5sum {}` berechnet für jede gefundene Datei einen 128-bit MD5-Hashwert

`2>/dev/null` evtl. Fehlermeldungen werden an das Nulldevice geleitet (virtuelle Gerätedatei, die alle Daten verwirft)

`sort` sortiert die Hashwerte für:

`uniq -w 32 -D` ermittelt doppelte Hashwerte (anhand der ersten 32 Zeichen; `-D` zeigt nur Duplikate)

Beispiel: PDF-Dokumente zusammenführen

- Zusammenführen, Zerteilen, Rotieren, ...

- `pdftk`
- `pdfsam`
- `pdfunite`
- `...`

Beispiel: PDF-Dokumente zusammenführen

```
# pdftk teil_1.pdf teil_2.pdf cat output alles.pdf
```

Beispiel: PDF-Dokumente zerteilen

```
# pdftk alles.pdf burst output teil_%02d.pdf
```

Beispiel: Dateien konvertieren

Audio-Dateien konvertieren mit `lame` (`for`-Schleife):

```
# for i in $(ls); do lame -r $i $i.wav; done
```

...für bestimmte Dateien & mit sauberer Endung:

```
# for i in $(find . -name "*.pcm"); \
do lame -m m $i ${i%.pcm}.wav; done
```

Audio-Dateien konvertieren mit `ffmpeg`:

```
# ffmpeg -b 192k -i ein aus.mp3
```

Video-Dateien konvertieren mit `ffmpeg`:

```
# ffmpeg -i Doing_Time.avi Doing_Vipassana.mp4
```

Beispiel: Zufallszahlen mit $RANDOM

- $RANDOM = interne Bash-Funktion, die eine (pseudo-)zufällige Zahl zwischen 0 und 32767 zurückgibt.

Zufallszahlen mit $RANDOM

```
# echo $RANDOM
13314
```

Zufallszahlen mit $RANDOM: Würfel-Version

```
# echo $[ $RANDOM % 6 + 1]
6
```

Beispiel: Skript für Zufallszahlen mit $RANDOM

Errechnet beliebige Anzahl von Zufallszahlen / Einmaligkeit

```
#!/bin/bash
echo "Wieviele ?"; read ANZAHL
ZAEHLER=1
> zufallszahlen.txt
while [ "$ZAEHLER" -le $ANZAHL ]
do
  ZAHL=$RANDOM
  echo $ZAHL >> zufallszahlen.txt
  ZAEHLER=$(($ZAEHLER + 1))
done
EINMALIG=$(sort -n zufallszahlen.txt | uniq | wc -l)
MEHRFACH=$(($ANZAHL - $EINMALIG))
ANTEIL=$(echo "scale=3; $MEHRFACH / $ANZAHL * 100" | bc)
echo "Bei insgesamt $ANZAHL mit 'RANDOM' erzeugten Zufallszahlen
sind $EINMALIG einmalig, d.h. es gab $MEHRFACH Mehrfachvorkommen
($ANTEIL %)."
```

„Russisch Roulette“ für Admins

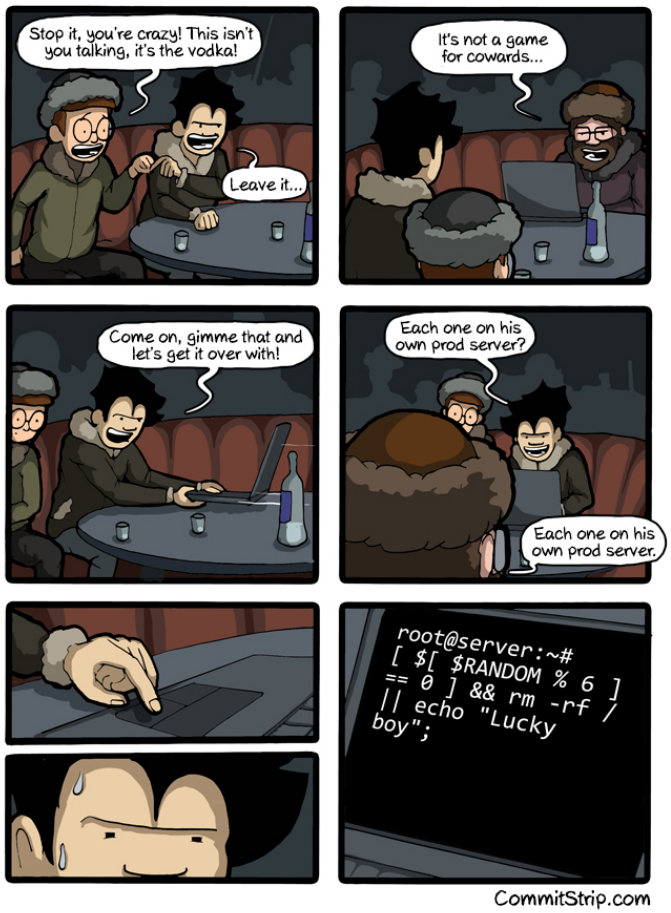


Abbildung : Quelle: CommitStrip.com

Beispiel: Zufallszahlen mit $RANDOM

Würfel (⇒ Ergebnisse zwischen 0 und 5)

```
# echo $[ $RANDOM % 6 ]
```

„Russisch Roulette“ für Admins (mit 5 von 6 Kugeln)

```
# [ $[ $RANDOM % 6 ] == 0 ] && rm -rf / || \
  echo “Glück gehabt!”
```

`[ $[ $RANDOM % 6 ] == 5 ]` Test, ob `$[ $RANDOM % 6 ]` = 5

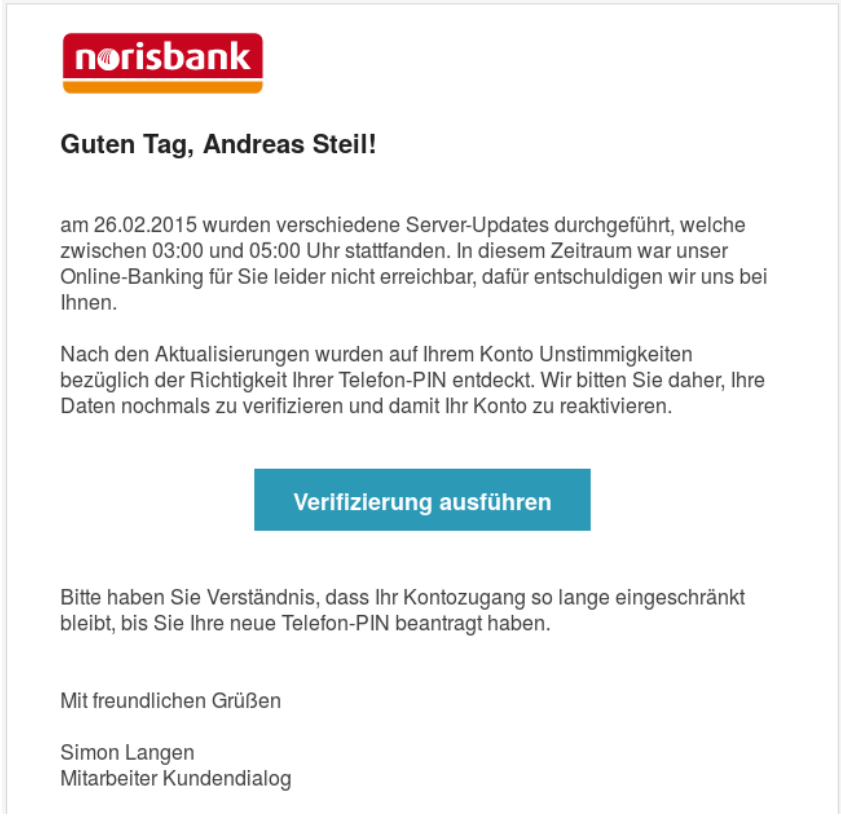
`&&` wenn voriger Befehl erfolgreich (Exit Code = 0 / `TRUE`)

`rm -rf /` lösche komplettes Wurzelverzeichnis (ohne Rückfrage!)

`||` wenn voriger Befehl nicht erfolgreich (Exit Code = 1 / `FALSE`)

`echo "Glück gehabt!"` Entwarnung ...

Beispiel: Spam or (Not) Spam 1/2



norisbank

Guten Tag, Andreas Steil!

am 26.02.2015 wurden verschiedene Server-Updates durchgeführt, welche zwischen 03:00 und 05:00 Uhr stattfanden. In diesem Zeitraum war unser Online-Banking für Sie leider nicht erreichbar, dafür entschuldigen wir uns bei Ihnen.

Nach den Aktualisierungen wurden auf Ihrem Konto Unstimmigkeiten bezüglich der Richtigkeit Ihrer Telefon-PIN entdeckt. Wir bitten Sie daher, Ihre Daten nochmals zu verifizieren und damit Ihr Konto zu reaktivieren.

Verifizierung ausführen

Bitte haben Sie Verständnis, dass Ihr Kontozugang so lange eingeschränkt bleibt, bis Sie Ihre neue Telefon-PIN beantragt haben.

Mit freundlichen Grüßen

Simon Langen
Mitarbeiter Kundendialog

URL des Links:

```
https://3c.gmx.net/mail/client/dereferrer?redirectUrl=http%3A%2F%2Fnoris-tele-verification.info \
%2Fbanking%2Fservices%2Fnorisbank-online%2...
```

Beispiel: Spam or (Not) Spam 2/2

Registrar ermitteln

```
# whois noris-tele-verification.info | grep Registrant
Registrant Name:Legato LLC
Registrant Street: Lesnaya 23, korpus 49
Registrant City:Samara
Registrant Postal Code:443002
Registrant Country:RU
[...]
Registrant Email:noris-tele-verification.info@allperson.ru
```

E-Mail-Adresse ermitteln ...

```
# whois noris-tele-verification.info | grep "Admin Email" | cut -d ":" -f 2
noris-tele-verification.info@allperson.ru
```

... E-Mail schreiben

```
# mail -s "Thanks for greetings..." $ADRESSE < echo "but: Don't bother me!"
```

Beispiel: Forkbombe für die Bash

- Forkbomb = Programm, das sich selbst durch rekursive Kopien unendlich vervielfältigt, um so alle verfügbaren Systemressourcen zu verbrauchen und das System zu blockieren

Forkbombe für die Bash (Normalform)

```
function f() {
    f | f&
}
f
```

Forkbombe für die Bash (Kurzform)

```
:(){ :|:& };:
```

Beispiel: Forkbombe für die Bash

Forkbombe für die Bash (Kurzform)

```
:(){ :|:& };:
```

Definition der Funktion »:«:

- `:()` definiert die Funktion »:« (immer wenn »:« aufgerufen wird, mach was in `{...}` steht)
- `:` neue Kopie von »:«ausführen
- `|` leiten die Standardausgabe um auf:
- `:` noch eine Kopie von »:«
- `&` im Hintergrund ausführen
- `;` beendet die Definition von »:«
- `:` Aufruf von »:«: setzt die Lawine in Gang ...

Beispiel: Suche Amelie

- `$ suche amelie`
  ⇒ Ergebnis:

Beispiel: Suche Amelie

Skript-Snippets:

```
#! /bin/bash
### TODO ... ;-)
[...]
for i in "$@";
do lynx -dump http://www.google.com/search?q="$@" > ergebnis;
echo $@ > suchbegriff;
done                                        # Ergebnis Bildersuche
[...]
grep ergebnis -i -A 3 ‘cat suchbegriff‘ \
       | head -3 | tail -1 | \
       cut -d\[ -f 2 | cut -d\] -f 1
join ...                                    # => URL des Bildes
[...]
bild = ... ‘convert ....jpg ....pgm‘        # Bild einschl. Umwandlung
                                            # in Rastergrafik zur ...
aview $bild -driver curses                  # ... Darstellung als ASCII-Art
[...]
```

Some Weblinks

GNU Bash – GNU-Projektseite zur Bash
: `http://www.gnu.org/software/bash/bash.html`

Bourne-Again SHell Manual `http://www.gnu.org/software/bash/manual/`
: „Offizielles“ Manual in mehreren Formaten (HTML, PDF, u.a.)

Bash Guide for Beginners `http://www.tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf`
: sehr ausführliche und fundierte Darstellung von Machtelt Garrels vom TLDP (PDF, englisch)

Netzmafia – Einführung in die Shell
: `http://www.netzmafia.de/skripten/unix/unix2.html`
sehr gute Einführung von Prof. Jürgen Plate von den grundlegenden Prinzipien (Metazeichen, Pipes, Ein-und Ausgabeumleitung) bis zum Skripting

Einführung in die Bourne Again Shell
: `http://www.selflinux.org/selflinux/pdf/bash_basic.pdf`
kleine Einführung des SelfLinux-Projekts mit einfachen Beispielen

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096