Bashscripting 101

Einführung in Bashscripten

von Marius Schwarz

für die (GNU) Linux UserGroup BraunSchweig

Stand Juli 2016

Bashscripting 101

- Protagonisten:

ls	Auflisten von Verzeichnissen
awk	zerlegt Textzeilen und baut sie neu
sed	Ersetzen von Texten
grep	Suchen von Texten in Dateien
cat	gibt Inhalte von Dateien aus.
sort	sortiert alle Eingaben und gibt diese wieder aus
rm	entfernt eine Datei
bash	öffnet eine neue Bashshell

- Ein- und Ausgaben

stdin	Eingabestrom
stdout	Ausgabestrom aka 1
stderr	Ausgabestrom für Fehlermeldungen aka 2

Alle vorgestellten Befehle haben eine Myriade an möglichen Einsatzzwecken und Optionen. Mit „man befehl“ kann man sich die Optionen ansehen und Informationen zum Befehl durchlesen : „man grep“

Bashscripting 101

Die Bashshell

Was ist eine Bashshell ?

Der Begriff Bashshell ist bereits ein zusammengesetztes Wort aus Bash und Shell.

Eine Shell ist ein Programm zum Ausführen von Befehlen auf einem Computer.

Bash ist der Name der wohl am meisten genutzten Shell.

Ein Terminal ist ein Fenster in dem üblicherweise eine Shell gestartet wird.

Neben der Bash-Shell gibt noch dutzende anderer Shells, die aber nicht ganz so verbreitet sind bspw. „sh“ „tsh“ „zsh“ usw. Das liegt u.a. daran, daß die Bash-Shell extrem mächtig ist.

Die Aufgabe der Shell ist es, neben dem reinen Ausführen der Befehle, auch die Ein- und Ausgaben an die Befehle möglich zu machen. Eine Shell ist NICHT zwangsweise nötig um ein Programm auszuführen. Es macht es nur einfacher für Menschen direkt mit Befehlen zu arbeiten.

Bashscripting 101

Die Bashshell

Ein leeres Terminalfenster in dem eine Bash gestartet wurde:

```
[marius@eve ~]$
```

Links befindet sich der sogenannte Prompt. Den Inhalt kann man anpassen, wenn man möchte. Üblicherweise steht dort der Benutzername, der Computername und der Pfad wo man ist.

Bashscripting 101

Die Bashshell

Die Bash-Shell kann sich selbst aufrufen:

```
[marius@eve ~]$ bash
[marius@eve ~]$
[marius@eve ~]$ ps auxf |grep bash
marius   21482  0.0  0.0 122664  4980 pts/2    Ss   17:54   0:00  \_ bash
marius   21933  0.5  0.0 122664  4848 pts/2    S    18:05   0:00      \_ bash
```

Obwohl keine besondere Ausgabe erfolgt, haben wir jetzt zwei Bashprogramm laufen, wie uns die Prozessliste (ps) zeigt, wenn wir nach bash filtern(grep)

Bashscripting 101

Die Bashshell

Durch Eingabe des Dateinamens des Befehls, den man ausführen will, wird das Programm gestartet:

```
[marius@eve ~]$ dnf search nemo
Letzte Prüfung auf abgelaufene Metadaten: vor 6 days, 6:17:21 am Fri Jul 22 11:51:00 2016.
======== N/S Treffer: nemo ========================================================
nemo.x86_64 : File manager for Cinnamon
nemo.i686 : File manager for Cinnamon
denemo-feta-fonts.noarch : Denemo feta fonts
nemo-emblems.noarch : Emblem support for nemo
...
```

Damit ein Programm gefunden wird, muß es sich sogenannten PFAD befinden. Dies ist Sammlung von Verzeichnissen in denen die Shell nach dem Programm suchen soll. So muß nicht immer die ganze Festplatte abgesucht werden bzw. der komplette Pfad angeben werden.
(DNF ist der Paketmanager von Fedora, NEMO ein Dateiverwaltungsprogramm)

Bashscripting 101

Grundlagen Ein- und Ausgaben

Jedes Programm, daß man startet, hat automatisch von Linux drei Ressourcen zum Einlesen und Ausgeben von Daten zugewiesen bekommen.

Jedes Programm, daß man in der Bash startet, hat automatisch von Linux drei Ressourcen zum Einlesen und Ausgeben von Daten zugewiesen bekommen, die per Pipe „Daten“ an andere in dieser Bashshell gestartete Programme übergeben können.

Bashscripting 101

Grundlagen Ein- und Ausgaben

Ziele dieser Ein- und Ausgaben können reale Dateien sein oder „virtuelle“ Laufwerke, die nur im Ram zur Laufzeit existieren.

Die Datenströme eines Programmes werden, wenn nicht anders angegeben, in der Bash immer vom oder an das „Fenster/Konsole/Terminal„ eingelesen/ausgegeben.

Die Datenströme können über die Steuerungszeichen „>“ „>>“ „<“ „<<“ „|“ (ALTGR + >) umgelenkt werden. Die „Pfeilspitze“ gibt die Richtung des Datenflußes an: „<“ Eingabe / „>“ Ausgabe .

Bashscripting 101

Grundlagen Ein- und Ausgaben

Es werden immer zwei Ausgabeströme erzeugt: stdout und stderr

Stdout ist für alle normalen Ausgaben eines Programmes zu benutzen.

Stderr dagegen, ist ausschließlich für Fehlermeldungen zu benutzen. Dies garantiert, daß Fehlermeldungen und Datenausgaben sich nicht vermischen und getrennt ausgewertet werden können.

Beispiel: ls > daten.txt 2 > error.log

Bashscripting 101

Grundlagen Ein- und Ausgaben

Beispiel: Ausgabe

```
[marius@eve test]$ ls
1bRZbG-0002Fo-Fx-D 1bScpn-0006pw-3s-D 1bSe0w-0001DP-Gw-D 1bSLEI-0007QR-CU-D 1bSMGq-0001nZ-W4-D
1bRZbG-0002Fo-Fx-H 1bScpn-0006pw-3s-H 1bSe0w-0001DP-Gw-H 1bSLEI-0007QR-CU-H 1bSMGq-0001nZ-W4-H
1bRZif-0003ca-Mc-D 1bScTu-0003eY-31-D 1bSeDf-0003Kp-4E-D 1bSLFH-0007op-SG-D 1bSMHm-0001tY-2w-D
1bRZif-0003ca-Mc-H 1bScTu-0003eY-31-H 1bSeDf-0003Kp-4E-H 1bSLFH-0007op-SG-H 1bSMHm-0001tY-2w-H
1bRZIT-0007lb-Es-D 1bScYx-0004LV-Du-D 1bSfBq-0003nZ-TL-D 1bSLiS-0004MM-AC-D 1bSNwM-0002Ru-Al-D
1bRZIT-0007lb-Es-H 1bScYx-0004LV-Du-H 1bSfBq-0003nZ-TL-H 1bSLiS-0004MM-AC-H 1bSNwM-0002Ru-Al-H
1bRZJa-0007sP-7J-D 1bScZj-0004gr-TH-D 1bSfdl-0000Tu-1T-D 1bSLk7-0004Yp-Ch-D 1bSUo2-0000Xi-CO-D
1bRZJa-0007sP-7J-H 1bScZj-0004gr-TH-H 1bSfdl-0000Tu-1T-H 1bSLk7-0004Yp-Ch-H 1bSUo2-0000Xi-CO-H
1bS1lB-00062T-MY-D 1bSdeY-000622-Sq-D 1bShlO-0006Jb-Lf-D 1bSLlW-0004gN-FO-D
1bS1lB-00062T-MY-H 1bSdeY-000622-Sq-H 1bShlO-0006Jb-Lf-H 1bSLlW-0004gN-FO-H


[marius@eve test]$ ls > dateiname.txt
[marius@eve test]$
```

Bashscripting 101

Grundlagen Ein- und Ausgaben

Lenkt man die Ausgabe auf eine noch nicht existierende Datei um, wird diese angelegt.

Lenkt man die Ausgabe auf eine existierende Datei, wird diese überschrieben.

Benutzt man zum Umlenken auf eine existierende Datei „>>“ statt „>“, wird die Ausgabe an die Datei hinten angefügt.

Benutzt man das „|“ Zeichen zwischen zwei Befehlen, wird die Ausgabe des ersten Programms an die Eingabe des zweiten Programms direkt umgeleitet.

Bashscripting 101

Grundlagen Ein- und Ausgaben

Beispiel: Eingabe

```
[marius@eve test]$ ls > dateiname.txt
[marius@eve test]$ cat < dateiname.txt
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-H
1bRepT-0004FX-4y-D
1bRepT-0004FX-4y-H
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-H
1bRYB8-00051u-BG-D
1bRYB8-00051u-BG-H
1bRYbI-0000j1-2O-D
1bRYbI-0000j1-2O-H
1bRYHm-0005wX-0I-D
1bRYHm-0005wX-0I-H
1bRYRO-0007aR-Gu-D
```

Bashscripting 101

Grundlagen Ein- und Ausgaben

Befehle wie „ls“ können Ihre Ausgabe so formatieren, daß diese optimal in das Fenster passen, in dem die Daten angezeigt werden.

Leitet man die Ausgabe in eine Datei um, geht dies nicht mehr.

Die Ausgabe von Befehlen wie „ls“, „grep“, „sed“, „awk“ erfolgt immer ZEILENWEISE wenn nicht anders gewollt.

Bashscripting 101

Grundlagen Ein- und Ausgaben

Benutzt man das „|“ Zeichen zwischen zwei Befehlen, wird die Ausgabe des ersten Programms an die Eingabe des zweiten Programms direkt umgeleitet.

Diese sogenannte PIPE (Tunnel) verändert die Ausgabe, da das ausgebende Programm die Ausgabe an die virtuellen Maße des Ausgabemediums (z.b. dem Fenster) anpassen kann. In einer PIPE geht das nicht, daher wird im PIPE-Beispiel die Ausgabe wieder zeilenweise gemacht.

Bashscripting 101

Grundlagen Ein- und Ausgaben

Beispiel: PIPE

```
[marius@eve test]$ ls | cat
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-H
1bRepT-0004FX-4y-D
1bRepT-0004FX-4y-H
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-H
1bRYB8-00051u-BG-D
1bRYB8-00051u-BG-H
1bRYbI-0000j1-2O-D
1bRYbI-0000j1-2O-H
1bRYHm-0005wX-0I-D
1bRYHm-0005wX-0I-H
1bRYRO-0007aR-Gu-D
...
```

Bashscripting 101

Die Aufgabe

„Lösche in einem Verzeichnis alle die Dateien, welche die Zeichenfolge zoho.com enthalten.“

Schwierigkeitsgrad 1

Bashscripting 101

Annahmen für diese Aufgabe :

- Verzeichnisname: test

- Wir befinden uns in dem Verzeichnis

- Wir nehmen an, daß es sich um den Zwischenspeicher eines Mailservers handelt, welcher den Header einer Email in der Datei „id-H“ und den eigentlichen Inhalt in der Datei „id-D“ ablegt.

Bashscripting 101

Inhalt des Verzeichnisses mit ls auflisten

```
[marius@eve test]$ ls
1bRaMt-0001Yr-7y-D 1bS1pr-0006zV-1W-D 1bSdgT-0006Ua-Kc-D 1bShwm-0008NB-Bc-D 1bSlqb-0006xF-TV-D
1bRaMt-0001Yr-7y-H 1bS1pr-0006zV-1W-H 1bSdgT-0006Ua-Kc-H 1bShwm-0008NB-Bc-H 1bSlqb-0006xF-TV-H
1bRepT-0004FX-4y-D 1bS3tT-00030n-8Y-D 1bSdiZ-0006hg-1c-D 1bShWV-0003V7-8u-D 1bSlqE-0006uj-2m-D
1bRepT-0004FX-4y-H 1bS3tT-00030n-8Y-H 1bSdiZ-0006hg-1c-H 1bShWV-0003V7-8u-H 1bSlqE-0006uj-2m-H
…
1bRZbG-0002Fo-Fx-D 1bScpn-0006pw-3s-D 1bSe0w-0001DP-Gw-D 1bSLEI-0007QR-CU-D 1bSMGq-0001nZ-W4-D
1bRZbG-0002Fo-Fx-H 1bScpn-0006pw-3s-H 1bSe0w-0001DP-Gw-H 1bSLEI-0007QR-CU-H 1bSMGq-0001nZ-W4-H
1bRZif-0003ca-Mc-D 1bScTu-0003eY-31-D 1bSeDf-0003Kp-4E-D 1bSLFH-0007op-SG-D 1bSMHm-0001tY-2w-D
1bRZif-0003ca-Mc-H 1bScTu-0003eY-31-H 1bSeDf-0003Kp-4E-H 1bSLFH-0007op-SG-H 1bSMHm-0001tY-2w-H
1bRZIT-0007lb-Es-D 1bScYx-0004LV-Du-D 1bSfBq-0003nZ-TL-D 1bSLiS-0004MM-AC-D 1bSNwM-0002Ru-Al-D
1bRZIT-0007lb-Es-H 1bScYx-0004LV-Du-H 1bSfBq-0003nZ-TL-H 1bSLiS-0004MM-AC-H 1bSNwM-0002Ru-Al-H
1bRZJa-0007sP-7J-D 1bScZj-0004gr-TH-D 1bSfdl-0000Tu-1T-D 1bSLk7-0004Yp-Ch-D 1bSUo2-0000Xi-CO-D
1bRZJa-0007sP-7J-H 1bScZj-0004gr-TH-H 1bSfdl-0000Tu-1T-H 1bSLk7-0004Yp-Ch-H 1bSUo2-0000Xi-CO-H
1bS1lB-00062T-MY-D 1bSdeY-000622-Sq-D 1bShlO-0006Jb-Lf-D 1bSLlW-0004gN-FO-D
1bS1lB-00062T-MY-H 1bSdeY-000622-Sq-H 1bShlO-0006Jb-Lf-H 1bSLlW-0004gN-FO-H
[marius@eve test]$
```

Bashscripting 101

Inhalt des Verzeichnisses mit ls -la auflisten

```
[marius@eve test]$ ls -la
insgesamt 1120
drwxrwxr-x.  2 marius marius  12288 28. Jul 16:10 .
drwx------. 91 marius marius  20480 28. Jul 16:11 ..
-rw-r-----.  1 marius marius   4019 28. Jul 16:08 1bRaMt-0001Yr-7y-D
-rw-r-----.  1 marius marius    733 28. Jul 16:08 1bRaMt-0001Yr-7y-H
-rw-r-----.  1 marius marius 108141 28. Jul 16:08 1bRepT-0004FX-4y-D
-rw-r-----.  1 marius marius    761 28. Jul 16:08 1bRepT-0004FX-4y-H
-rw-r-----.  1 marius marius   4669 28. Jul 16:08 1bRY9X-0004sP-PK-D
...
-rw-r-----.  1 marius marius    748 28. Jul 16:08 1bSLrf-0005dj-Qx-H
-rw-r-----.  1 marius marius   4636 28. Jul 16:08 1bSLrv-0005yp-6A-D
-rw-r-----.  1 marius marius    744 28. Jul 16:08 1bSLrv-0005yp-6A-H
-rw-r-----.  1 marius marius   4611 28. Jul 16:08 1bSLuh-0006Jp-OW-D
-rw-r-----.  1 marius marius    738 28. Jul 16:08 1bSLuh-0006Jp-OW-H
-rw-r-----.  1 marius marius   4635 28. Jul 16:08 1bSLzY-00079v-5W-D
-rw-r-----.  1 marius marius    738 28. Jul 16:08 1bSLzY-00079v-5W-H
-rw-r-----.  1 marius marius   3559 28. Jul 16:08 1bSM2S-0007nc-Kr-D
[marius@eve test]$
```

Bashscripting 101

Schritt 1 – Die Dateien durchsuchen

grep -i 'zoho.com' *

Die Option „-i“ schaltet die Unterscheidung von Groß- und Kleinbuchstaben ab: Zoho.com = zoho.com = ZOHo.CoM usw.

Um ALLE Dateien eines Verzeichnisses zu durchsuchen, geben wir als Dateinamen „*“ an. Ansonsten steht dort der Dateiname :

grep -i „Hallo“ /home/marius/.welcome.txt

Oder auch per PIPE

cat /home/marius/welcome.txt | grep -i „Hallo“

Bashscripting 101

Beispielausgabe: grep

```
[marius@eve test]$ grep -i zoho.com *
1bRaMt-0001Yr-7y-D:Return-path: <paolomsin@zoho.com>
1bRaMt-0001Yr-7y-D:    (envelope-from <paolomsin@zoho.com>)
1bRaMt-0001Yr-7y-D:From: "Rainer Beck" <paolomsin@zoho.com>
1bRaMt-0001Yr-7y-D:Reply-To: paolomsin@zoho.com
1bRaMt-0001Yr-7y-D:                                (paolomsin[at]zoho.com)
1bRaMt-0001Yr-7y-H:paolomsin@zoho.com
1bRaMt-0001Yr-7y-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 09:32:03 +0200
1bRaMt-0001Yr-7y-H:023T To: paolomsin@zoho.com
1bRY9X-0004sP-PK-D:Return-path: <paolomsin@zoho.com>
1bRY9X-0004sP-PK-D:    (envelope-from <paolomsin@zoho.com>)
1bRY9X-0004sP-PK-D:From: "Rainer Beck" <paolomsin@zoho.com>
1bRY9X-0004sP-PK-D:Reply-To: paolomsin@zoho.com
1bRY9X-0004sP-PK-D:                                (paolomsin[at]zoho.com)
1bRY9X-0004sP-PK-H:paolomsin@zoho.com
1bRY9X-0004sP-PK-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 07:10:07 +0200
1bRY9X-0004sP-PK-H:023T To: paolomsin@zoho.com
```

Bashscripting 101

Bewertung der „grep“ Ausgabe

Der Befehl „grep“ sucht in Dateien oder in seinem stdin nach einer Zeichenfolge.

Sobald er mehr als eine Datei durchsuchen soll, gibt „grep“ mit aus, in welcher Datei er die Zeichenfolge gefunden hat.

Beispiel: grep -i „zoho.com *

Ergebnis: „1bSHZX-0002f6-WF-H:lusjiaying@zoho.com“

Vor dem „:“ steht der Dateiname, danach der Inhalt.

Für unsere Beispielaufgabe brauchen wir den Inhalt nicht, es reicht uns, daß es drinsteht. Daher müssen wir die Ausgabe mit dem Befehl „sed“ vom Inhalt befreien. Dazu setzen wir einen regulären Ausdruck ein, der ab einer bestimmten Zeichenfolge alles nachfolgende löscht. (mit „“ (nichts) ersetzt)

Bashscripting 101

Schritt 2 - Processing mit dem „sed“ - Streameditor

```
[marius@eve test]$ grep -i zoho.com * | sed -e "s/:.*$//"
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-D
1bRaMt-0001Yr-7y-H
1bRaMt-0001Yr-7y-H
1bRaMt-0001Yr-7y-H
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-D
1bRY9X-0004sP-PK-H
1bRY9X-0004sP-PK-H
1bRY9X-0004sP-PK-H
```

Bashscripting 101

Processing mit dem „sed“ - Streameditor

```
[marius@eve test]$ grep –i zoho.com * | sed –e "s/:.*$//"
```

Wir setzen die PIPE ein um „grep“ und „sed“ zu verbinden.

Die Option „-e“ meint Ausführen/Execute das in Anführungszeichen stehende wilde Zeichenkonstrukt ist der reguläre Ausdruck, der speziell für unser Beispiel alles (.*) ab/inclusive dem „:“ bis zum Zeilenende ($) durch „nichts“ => „“ substituiert(ersetzt) (s/).

Reguläre Ausdrücke sind cool, aber heftig, ihr braucht ihn hier nur zur Kenntnis nehmen, nicht gleich verstehen.

Bashscripting 101

Schritt 3 - Processing mit „awk“

Als Ergebnis bleiben nur noch die gewünschten Dateinamen übrig. Damit können wir nun den nächsten Schritt machen und uns eine Anweisung bauen, welche die *jeweilige* Datei löscht.

Dazu setzen wir als nächstes „awk“ ein. Dieses Programm nimmt eine Eingabezeile, zerlegt es in seine Einzelteile in dem „normalerweise“ an allen Stellen getrennt wird, an der ein Leerzeichen steht. Alle „Teile“ der bisherigen Eingabezeile werden durchnummeriert und können beliebig neu zusammen gesetzt werden.

In unserem Beispiel haben wir nur ein Teil, dessen Variable ist : $1

Bashscripting 101

Processing mit „awk“

Beispiel: „Ein Bauer sucht Frau“

Dies wird so „zerlegt“ :

$1 = Ein
$2 = Bauer
$3 = sucht
$4 = Frau

```
[marius@eve test]$ echo "Ein Bauer sucht Frau" | awk '{print $1" "$4" "$3" "$2;}'
Ein Frau sucht Bauer
[marius@eve test]$ echo "Ein Bauer sucht Frau" | awk '{print "Eine "$4" "$3" einen "$2;}'
Eine Frau sucht einen Bauer
```

Man kann also Ausgaben von Programmen per Pipe an „awk“ beliebig aufbereiten, wenn man möchte.

Bashscripting 101

Processing mit „awk“

```
[marius@eve test]$ grep -i zoho.com * | sed -e "s/:.*$//" | awk '{print "rm -f "$1;}'
rm -f 1bRaMt-0001Yr-7y-D
rm -f 1bRaMt-0001Yr-7y-D
rm -f 1bRaMt-0001Yr-7y-D
rm -f 1bRaMt-0001Yr-7y-D
rm -f 1bRaMt-0001Yr-7y-D
rm -f 1bRaMt-0001Yr-7y-H
rm -f 1bRaMt-0001Yr-7y-H
rm -f 1bRaMt-0001Yr-7y-H
rm -f 1bRY9X-0004sP-PK-D
rm -f 1bRY9X-0004sP-PK-D
rm -f 1bRY9X-0004sP-PK-D
rm -f 1bRY9X-0004sP-PK-D
rm -f 1bRY9X-0004sP-PK-D
```

Bashscripting 101

Processing mit „awk“

```
[marius@eve test]$ grep -i zoho.com * | sed -e "s/:.*$//" | awk '{print "rm -f "$1;}'
rm -f 1bRaMt-0001Yr-7y-D
…
```

In unserem Fall, hatten wir nur den Dateinamen in der Variablen $1 und bauen uns nun damit einen Löschbefehl zusammen.

„rm“ mit der Option „-f“ löscht ohne nachzufragen die dahinter angegebene Datei.

```
rm -f 1bRaMt-0001Yr-7y-D
```

NOCH ist das ganze aber nur eine Ausgabe wie jede andere auch. Es reicht also nicht „rm -f ….“ auszugeben, wir müssen es noch ausführen.

Bashscripting 101

Schritt 4 - Befehle mit Bash ausführen

```
[marius@eve test]$ grep -i zoho.com * | sed -e "s/:.*$//" | awk '{print "rm -vf "$1;}' | bash
„1bRaMt-0001Yr-7y-D" wurde entfernt
„1bRaMt-0001Yr-7y-H" wurde entfernt
„1bRY9X-0004sP-PK-D" wurde entfernt
„1bRY9X-0004sP-PK-H" wurde entfernt
„1bRYB8-00051u-BG-D" wurde entfernt
„1bRYB8-00051u-BG-H" wurde entfernt
„1bRYbI-0000j1-2O-D" wurde entfernt
„1bRYbI-0000j1-2O-H" wurde entfernt
„1bRYHm-0005wX-0I-D" wurde entfernt
„1bRYHm-0005wX-0I-H" wurde entfernt
„1bRYRO-0007aR-Gu-D" wurde entfernt
„1bRYRO-0007aR-Gu-H" wurde entfernt
„1bRZ89-0006D6-Qq-D" wurde entfernt
„1bRZ89-0006D6-Qq-H" wurde entfernt
„1bRZ90-0006IK-8n-D" wurde entfernt
„1bRZ90-0006IK-8n-H" wurde entfernt
„1bRZbG-0002Fo-Fx-D" wurde entfernt
„1bRZbG-0002Fo-Fx-H" wurde entfernt
„1bRZif-0003ca-Mc-D" wurde entfernt
```

Bashscripting 101

Bewertung der ersten Aufgabe

Die Anweisung die wir gebaut haben:

```
grep -i zoho.com * | sed -e "s/:.*$//" | awk '{print "rm -vf "$1;}' | bash
```

- sucht also nach dem Text „zoho.com“ und gibt Dateiname und Fund aus
- ersetzt den Fund mit „nichts“ (löscht den Text also)
- baut den übriggebliebenen Dateinamen in eine Löschanweisung ein
- und führt am Ende diese Löschanweisung per Bash aus

Die gestellte Aufgabe ist damit gelöst.

Die Lösung kann aber noch etwas verbessert werden, da logische Fehler enthalten sind. Z.B. will der Befehl bereits gelöschte Datei nochmals löschen, weil Mehrfachfunde vorkamen.

Bashscripting 101

Optimierung der ersten Aufgabe

Die Anweisung „grep -i zoho.com *“ findet zoho.com ggf. mehrfach in einem Textfile, da jeder Fund in einer neuen Zeile ausgegeben wird, in der auch wieder der Dateiname steht, führt dies am Ende zu doppelt und dreifachen Namen, die alle untereinander stehen.

Der „sort“ Befehl sort nun die Ausgabe von „sed“ um alphabetisch und wirft doppelte Dateinamen raus.

```
grep -i zoho.com * | sed -e "s/:.*$//" | sort -u | awk '{print "rm -vf "$1;}' | bash
```

Damit wird jetzt jede Datei genau einmal gelöscht.

Die doppelte Ausgabe ist für die gestellte 1. Aufgabe nicht tragisch, aber der logische Fehler führt bei anderen Anwendungen des gleichen Prinzips möglicherweise zu schwerwiegenden Fehlern.

Bashscripting 101

Die 2. Aufgabe

„Lösche in einem Verzeichnis alle die zusammenhängen Header- und Datendateien (-H und -D), welche im Header die Zeichenfolge zoho.com enthalten. Ignoriere Funde in der -D Datei“

Schwierigkeitsgrad 2

Bashscripting 101

Beispielausgabe: grep

```
[marius@eve test]$ grep -i zoho.com *-H
1bRaMt-0001Yr-7y-H:paolomsin@zoho.com
1bRaMt-0001Yr-7y-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 09:32:03 +0200
1bRaMt-0001Yr-7y-H:023T To: paolomsin@zoho.com
1bRY9X-0004sP-PK-H:paolomsin@zoho.com
1bRY9X-0004sP-PK-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 07:10:07 +0200
1bRY9X-0004sP-PK-H:023T To: paolomsin@zoho.com
1bRYB8-00051u-BG-H:paolomsin@zoho.com
1bRYB8-00051u-BG-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 07:11:46 +0200
1bRYB8-00051u-BG-H:023T To: paolomsin@zoho.com
1bRYbI-0000j1-2O-H:paolomsin@zoho.com
1bRYbI-0000j1-2O-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 07:38:48 +0200
1bRYbI-0000j1-2O-H:023T To: paolomsin@zoho.com
1bRYHm-0005wX-0I-H:paolomsin@zoho.com
1bRYHm-0005wX-0I-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 07:18:38 +0200
1bRYHm-0005wX-0I-H:023T To: paolomsin@zoho.com
1bRYRO-0007aR-Gu-H:paolomsin@zoho.com
1bRYRO-0007aR-Gu-H:    for paolomsin@zoho.com; Mon, 25 Jul 2016 07:28:34 +0200
1bRYRO-0007aR-Gu-H:023T To: paolomsin@zoho.com
1bRZ89-0006D6-Qq-H:paolomsin@zoho.com
```

Bashscripting 101

„grep“ mit Teilnamen benutzen

Wir hatten in der ersten Aufgabe gesagt, daß „*“ benutzt wird um alle Dateien anzusprechen. Das stimmt zwar, wir der Sache aber nicht ganz gerecht.

Tatsächlich wird jeder Dateiname im aktuellen Verzeichnis per regulären Ausdruck (siehe SED) gegen „*“ geprüft. „*“ bedeutet eigentlich „alle Zeichen – egal wie oft und in welcher Kombination“

Wir erweitern dies jetzt um nur noch Teile des Dateinamens zuberücksichtigen:

grep -i zoho.com *-H

Bedeutet, daß der Name der Datei „{egalwas}-H“ sein muß, also auf -H endet. Das schließt Dateien die auf „-D“ enden aus, was Teil der Aufgabe ist.

Bashscripting 101

„sed“ Ausdruck erweitern

Wir haben jetzt nur noch Dateinamen die auf -H enden. Die Aufgabe sagt aber, daß wird auch alle Dateien löschen sollen, die die gleiche ID haben, aber auf -D enden, weil diese zusammen gehören (Header und Daten)

Den Sed-Ausdruck müssen wir also um -H erweitern, so daß nur noch der Basisname(ID) übrigbleibt:

```
[marius@eve test]$  grep -i zoho.com *-H | sed -e "s/-H:.*$//"
1bRaMt-0001Yr-7y
1bRaMt-0001Yr-7y
1bRaMt-0001Yr-7y
1bRY9X-0004sP-PK
1bRY9X-0004sP-PK
1bRY9X-0004sP-PK
```

Dies filtern wir wieder mit „sort -u“ um alle doppelten zu entfernen und bauen uns einen neuen rm -f Befehl, der um „-*“ erweitert wurde, weil er ja „Basisname-*“ löschen soll.

Bashscripting 101

Neue Ausgabe vom Löschbefehl

```
[marius@eve test]$ grep -i zoho.com *-H | sed -e "s/-H:.*$//" | sort -u | awk '{print "rm -vf "$1"-*";}'
rm -vf 1bRaMt-0001Yr-7y-*
rm -vf 1bRY9X-0004sP-PK-*
rm -vf 1bRYB8-00051u-BG-*
rm -vf 1bRYbI-0000j1-2O-*
rm -vf 1bRYHm-0005wX-0I-*
rm -vf 1bRYRO-0007aR-Gu-*
rm -vf 1bRZ89-0006D6-Qq-*
rm -vf 1bRZ90-0006IK-8n-*
rm -vf 1bRZbG-0002Fo-Fx-*
rm -vf 1bRZif-0003ca-Mc-*
…
```

Das muß jetzt wieder an Bash gepiped werden, dann sind wir fertig.

```
[marius@eve test]$ grep -i zoho.com *-H | sed -e "s/-H:.*$//" | sort -u | awk '{print
"rm -vf "$1"-*";}' | bash
```

Bashscripting 101

Aufgabe 3 – Ein Bashscript daraus machen

Das machen wir nächste Woche :)