HOWTO zur Shee-Bang Zeile in Skript-Dateien

(C) 2006-2013 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: shell-shee-bang-HOWTO.txt,v 1.5 2013/03/25 11:52:11 tsbirn Exp $

Dieses Dokument beschreibt die Bedeutung der Shee-Bang Zeile in ausführbaren Dateien bzw. Skripten.

INHALTSVERZEICHNIS

---

1) Einführung

Die Shee-Bang Zeile bestimmt, mit welchem Programm (Interpreter) eine Skript-Datei ausgeführt wird. Fehlt diese Zeile, wird die Skript-Datei automatisch mit der AKTUELLEN Shell ausgeführt, in der das Kommando zum Start des Skriptes ausgelöst wurde (d.h. unter Linux normalerweise der "bash").

Der Einsatz des Shee-Bang Konzepts ist nicht auf Shell-Skripte beschränkt, sondern kann auf jede Art von Skriptsprache angewendet werden. Neben den klassischen Shells "sh", "csh", "tcsh", "ksh", "bash" und "zsh" gibt es noch viele andere Skript-Programmiersprachen (z.B. "sed", "awk", "perl", "php", "python", "ruby") die eine Skript-Datei ausführen können.

Ebenso können auch alle anderen Kommandos in einer Shee-Bang Zeile eingesetzt werden, die ihre Eingabe von einer Datei lesen können (z.B. "grep", "telnet", "ftp").

2) Syntax

Syntax der Shee-Bang Zeile: In der Skript-Datei sind in der 1. Zeile ab der 1. Spalte die beiden Zeichen "#!" (umgangssprachlich "Shee" und "Bang") und danach der ABSOLUTE (vollständige) Pfad zum Programm anzugeben, mit dem das Skript in der Skript-Datei ausgeführt werden soll:

```
vim test.sh
+----------------+
|#!/bin/bash     |
|...             |
+----------------+
```

Die Länge der Shee-Bang Zeile ist beschränkt (32, 64 oder 128 Zeichen je nach Linux-Version), weiterhin darf sie durch Leerraum getrennt maximal EIN zusätzliches Argument nach dem Programmpfad enthalten (hier "-u"):

```
vim test.sh
+----------------+
|#!/bin/bash -u  |
|...             |
+----------------+
```

Außer zur Trennung von Programmpfad und evtl. vorhandenem Argument darf die Shee-Bang Zeile keine Leerzeichen oder sonstigen Zeichen enthalten und muss wie unter Linux üblich mit einem "\n" abgeschlossen sein. Insbesondere andere Zeilenende-Formate (z.B. "\r\n" bei Windows und "\r" bei MacOS X) sind nicht erlaubt und führen zu Fehlern (Pfad wird nicht gefunden):

Sobald eine Skript-Datei ausführbar gemacht wurde, kann sie unter ihrem Namen als Kommando aufgerufen werden.

```
chmod +x skript.sh   # Ausführbar machen
./skript.sh          # Aufrufen
```

Eine Shee-Bang Zeile ab dem 2. Zeichen oder der 2.Zeile einer Skript-Datei wird vom Linux-Kernel nicht beachtet und ebenso vom ausführenden Interpreter als Kommentarzeile ignoriert.

3) Ablauf

```
---------
Der Linux-Kern liest den Inhalt der ausführbaren Skript-Datei nach ihrem Aufruf
ein, erkennt die beiden Zeichen "#!" als ERSTE Zeichen der Datei und
interpretiert daher den Rest der 1. Zeile als absoluten Pfad zu einem Kommando,
das er startet und ihm dabei die GESAMTE zum Aufruf verwendete Kommandozeile
als Argument mitgibt:

  /bin/bash ./test.sh
  <-------> <------->
  Shee-Bang Kmdozeile

Hierdurch wird hier eine (Sub)Shell "/bin/bash" gestartet, die das Skript
"test.sh" ausführt. Die ursprüngliche Shee-Bang Zeile --- die diesen Ablauf
auslöst --- wird dabei vom Skript-Interpreter (hier "bash") als KOMMENTARZEILE
interpretiert und überlesen, d.h. ignoriert. Dies ist der Grund, warum alle
Skriptsprachen unter Linux das Zeichen "#" als Beginn einer Kommentarzeile
interpretieren.

Enthält die Shee-Bang Zeile ein zusätzliches Argument, dann wird dieses vom
Linux-Kern mit in den generierten Aufruf eingesetzt:

  vim test2.sh
  +---------------------+
  |#!/usr/bin/bash -ivu |
  |...                  |
  +---------------------+
  chmod +x test2.sh
  ./test2.sh

Obiger Ablauf führt letztendlich zur Ausführung des folgenden Kommandos:

  /usr/bin/bash -ivu ./test2.sh
  <----------------> <-------->
      Shee-Bang       Kmdozeile

Der Linux-Kern kopiert alle Zeichen der Shee-Bang Zeile direkt (ohne
Interpretation durch eine Shell), d.h. es bedarf keiner Quotierung mit "...",
'...' und "\", um Sonderzeichen zu schützen.


4) Fehler
---------
Bei ungültigem Pfad (z.B. führender "/" vergessen) erfolgt beim Aufruf des
Skripts eine Fehlermeldung der ausführenden Shell (hier der bash):

  bash: ./tippfehler.sh: bin/sh: bad interpreter: file or directory not found

Beispiele für ungültige Pfade in der Shee-Bang Zeile:

  +---------------------+
  |#! /bin/sh           |       # Leerzeichen im Pfad
  |echo "Hallo Welt!"   |
  +---------------------+

  +---------------------+
  |#!bin/sh             |       # Pfad "bin/sh" gibt es nicht
  |echo "Hallo Welt!"   |
  +---------------------+

  +---------------------+
  |#!/bin/sh^M          |       # Strg-M (DOS/Windows-Zeilenende) im Pfad
  |echo "Hallo Welt!"   |
  +---------------------+

  +---------------------+
  |#!/bin/xyzabc123     |       # Kommando "xyzabc123" gibt es nicht
  |echo "Hallo Welt!"   |
  +---------------------+


5) Beispiele
------------
a) Ein (klassisches) Bourne-Shell-Skript "skript.sh" beginnt mit "#!/bin/sh":

  vim skript.sh
  +---------------------+
  |#!/bin/sh            |
  |echo "Hallo Welt!"   |
  +---------------------+
  chmod +x skript.sh
  ./skript.sh                   # --> /bin/sh ./skript.sh
```

b) Ein Z-Shell-Skript "zsh.sh" beginnt mit "#!/usr/bin/zsh":

```
  vim zsh.sh
  +---------------------+
  |#!/usr/bin/zsh       |
  |echo "Hallo Welt!"   |
  +---------------------+
  chmod +x zsh.sh
  ./zsh.sh                      # --> /usr/bin/zsh ./skript.sh
```

c) Ein Perl-Skript "hello.pl" beginnt mit "#!/usr/bin/perl -w":

```
  vim hello.pl
  +-------------------------+
  |#!/usr/bin/perl -w       |   # -w schaltet Warnungen ein
  |print "Hallo Welt!\n";   |
  +-------------------------+
  chmod +x hello.pl
  ./hello.pl                    # --> /usr/bin/perl -w ./hello.pl
```

d) Ein grep-Skript "made.grep" beginnt mit "#!/bin/grep REGEXP", ein Kommando das den Inhalt des Skripts nach Zeilen durchsucht, die zum regulären Ausdruck "REGEXP" passen. Bitte nicht in Anführungszeichen setzen, da diese sonst mitgesucht werden. Die Shell sieht diese Kommandozeile nämlich gar nicht, sondern nur der Linux-Kern!:

```
  vim made.grep
  +-----------------------------------+
  |#!/bin/grep made                   |
  |echo "Dies ist ein Text mit made"  |
  |ohne ... ohne                      |
  |mit made mit made mit made         |
  |ohne ... ohne                      |
  +-----------------------------------+
  chmod +x made.grep
  ./made.grep                   # --> /bin/made.grep made ./made.grep
```

e) Ein sed-Skript "vokal.sed" beginnt mit "#!/bin/sed -f" (sed = stream editor, programmierbarer Editor zum automatisierten Bearbeiten von Dateien). Die Option "-f" (file) ist notwendig, damit der sed-Interpreter das Skript aus einer Datei liest.

```
  vim vokal.sed
  +-------------------------------------------------------------+
  |#!/bin/sed -f                                                |
  |# Ersetzt alle Vokale durch nichts (s=substitute, g=global) |
  |s/[aeiou]//g                                                 |
  |s/[AEIOU]//g                                                 |
  +-------------------------------------------------------------+
  chmod +x vokal.sed
  vokal.sed < INPUT > OUTPUT    # --> /bin/sed -f ./vokal.sed < INPUT > OUTPUT
```

d) Ein awk-Skript "wc.awk" beginnt mit "#!/usr/bin/awk -f" (awk = Vorläufer von Perl). Der Pfad zum awk-Interpreter kann unterschiedlich lauten (z.B. auch "/usr/local/bin/gawk"), die Option "-f" (file) ist notwendig, damit der awk-Interpreter das Skript aus einer Datei liest.

```
  vim wc.awk
  +-------------------------------------------------------------+
  |#!/usr/bin/awk -f                                            |
  |# (programmiert den Linux-Befehl "wc" (word count) nach)     |
  |# Zählt Zeilen, Worte und Zeichen einer Datei                |
  |{                                                            |
  |    words += NF                                              |
  |    chars += length($0)                                      |
  |}                                                            |
  |END {                                                        |
  |    print NR, words, chars+NR                                |
  |}                                                            |
  +-------------------------------------------------------------+
  chmod +x wc.awk
  ./wc.awk INPUT...              # --> /usr/bin/awk -f ./wc.awk INPUT...
```